Universidade Federal Fluminense

LEANDRO FERNANDES DOS SANTOS

Método Dual Simplex para problemas canalizados com estruturas de dados eficientes

VOLTA REDONDA

2020

LEANDRO FERNANDES DOS SANTOS

Método Dual Simplex para problemas canalizados com estruturas de dados eficientes

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Otimização e Pesquisa Operacional.

Orientador:

Prof. D.Sc. Ricardo Silveira Sousa

Coorientador:

Prof. D.Sc. Wagner Rambaldi Telles

Universidade Federal Fluminense

VOLTA REDONDA

2020

Ficha catalográfica automática - SDC/BEM
Gerada com informações fornecidas pelo autor

S237m Santos, Leandro Fernandes dos
Método Dual Simplex para problemas canalizados com estruturas de dados eficientes / Leandro Fernandes dos Santos ; Ricardo Silveira Sousa, orientador ; Wagner Rambaldi Telles, coorientador. Volta Redonda, 2020.
120 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Volta Redonda, 2020.

DOI: http://dx.doi.org/10.22409/PPG-MCCT.2020.m.08828102608

1. Pesquisa Operacional. 2. Programação Linear. 3. Método Dual Simplex. 4. Estruturas de Dados. 5. Produção intelectual. I. Sousa, Ricardo Silveira, orientador. II. Telles, Wagner Rambaldi, coorientador. III. Universidade Federal Fluminense. Escola de Engenharia Industrial e Metalúrgica de Volta Redonda. IV. Título.

CDD -

Bibliotecária responsável: Andreia de Oliveira Paim - CRB7/5183

Método dual simplex para problemas canalizados com estruturas de dados eficientes

Leandro Fernandes dos Santos

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Aprovada por:

Ricardo Silveira Sousa

Prof. Ricardo Silveira Sousa, D.Sc.
INFES-UFF(Presidente)

Thiago Jordem Pereira

Prof. Thiago Jordem Pereira, D.Sc.
INFES-UFF

Prof. Nelson Machado Barbosa, D.Sc.
LCMAT-UENF

Volta Redonda, 19 de Fevereiro de 2020.

Dedicatória.

À minha querida esposa e família, presentes de Deus.

Agradecimentos

Primeiramente agradeço a Deus pela vida e por manter em mim viva a certeza de que a caminhada seria longa, mas a vitória certa. Sou grato a Ele por estar comigo em todos os momentos e por permitir a conclusão de mais esta etapa.

À minha amada esposa Ludimila Pimenta Alves Fernandes, pelo apoio, paciência e compreensão. Por todo o zelo que tens pelo nosso Amor. Eu Te Amo.

Aos meus pais, Getulio Fernandes dos Santos e Suely Aparecida Vieira Fernandes, meus maiores exemplos de caráter e boa conduta, e toda a minha família. Vocês são presentes de valor incalculável em minha vida.

Ao meu orientador, Prof. D.Sc. Ricardo Silveira Sousa, por todo apoio e empenho para que este trabalho se tornasse concreto. Obrigado pelos ensinamentos e conversas sempre proveitosas nas inúmeras reuniões que tivemos durante a realização deste trabalho.

Aos demais professores do MCCT do INFES-UFF. Todo conhecimento transmitido foi muito além das dimensões da sala de aula, sou imensamente grato.

Aos amigos do mestrado pela boa convivência e momentos de estudo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil - (CAPES). Código de Financiamento 001.

Resumo

Implementações eficientes e estáveis de métodos tipo simplex constituem um desafio, pois em primeira instância, exige um bom conhecimento do método, por conseguinte necessita-se realizar uma implementação que tire proveito de características presentes em problemas de grande porte, tais como a esparsidade. Sob esta ótica, a escolha correta das estruturas de dados, formas de atualização da base, tratamento numérico, entre outras, são essenciais para se obter uma implementação robusta.

No presente trabalho foi estudado e implementado o método Dual Simplex especializado em problemas de programação linear canalizados utilizando diferentes estruturas de dados a fim de avaliar o tempo computacional para cada uma delas. Foram utilizadas duas formas de atualização da base: Forma Revisada e Forma Produto da Inversa. Para cada uma das formas, testou-se três diferentes estruturas de dados para o armazenamento e manipulação das matrizes envolvidas: Forma densa, tratada aqui como Forma Indexada, Listas Encadeadas e conjunto de vetores compactos CSC/CSR. Também foi dada ênfase na forma de tratamento dos dados de entrada de cada problema testado, visando obter uma representação homogênea destes.

O Método Dual Simplex juntamente com as diferentes estruturas de dados e formas de atualização da base foi testado em três classes de problemas distintas: Problemas de mistura, oriundos da fabricação de rações, problemas criados por um gerador pseudo-aleatório de modelos de produção com alta esparsidade e instâncias do repositório da Netlib. Para alguns casos, os resultados obtidos foram comparados com a implementação do Dual Simplex contida no Solver CPLEX 12.8.

Ao realizar a implementação do método juntamente com a Forma Revisada e fazendo-se o uso de conjunto de vetores compactos CSC/CSR, as rotinas utilizadas nesta abordagem demandaram a construção de um algoritmo específico para utilização destes conjuntos de vetores, constituindo também uma das contribuições do trabalho.

Os resultados computacionais evidenciam a importância da utilização de estruturas de dados eficientes nas rotinas do método Dual Simplex canalizado, sendo que uma das melhores abordagens consiste no conjunto de vetores compactos CSC/CSR juntamente com a Forma Produto da Inversa, contudo para algumas classes de problemas, a implementação com listas encadeadas também se mostrou promissora.

Abstract

Efficient and stable implementations of simplex methods are challenging because, in the first instance, it requires good knowledge of the method, so an implementation that takes advantage of features present in large real problems such as sparsity needs to be implemented. From this point of view, the correct choice of data structures, forms of base updating, numerical treatment, among others, are essential to achieve a robust implementation.

In the present work, the Dual Simplex method specialized in bounded linear programming problems using different data structures was studied and implemented in order to evaluate the computational time for each one. Two forms of base update were performed, Revised Form and Product Form of the Inverse. In each form, we tried to test three different data structures for storing and manipulating the constraint matrix (A matrix) and the base inverse matrix: Dense form, treated here as Indexed, Linked Lists, and CSC/CSR compact vector set. Emphasis was also given to the threatment of input data, with the intention of obtaining a homogeneous representation of these.

The Dual Simplex Method with the different data structures and base update forms has been tested on three distinct problem classes: Mixing production problems, from a ration factory, instances set generated by a pseudo-random generator of high-sparsity production models, and instances of the Netlib repository. For validation purposes, the results obtained were compared with the Dual Simplex implementation contained in Solver CPLEX 12.8.

When implementing the method together with the Revised Form and using the CSC/CSR compact vector set, the routines used in this approach demanded the construction of a specific algorithm for the use of these vector sets, also constituting one of the contributions of the work.

The computational results show that one of the best approaches is the set of CSC/CSR compact vectors when used together with the Product Form of the Inverse approach, however for some problem classes, the implementation with linked lists is also promising.

Palavras-chave

1. Pesquisa Operacional
2. Programação Linear
3. Algoritmo Simplex
4. Método Dual Simplex
5. Modelagem Matemática
6. Esparsidade
7. Problemas Canalizados
8. Estabilidade Numérica
9. Estruturas de Dados

Glossário

PO	:	Pesquisa Operacional
PL	:	Programação Linear
DSC	:	Método Dual Simplex Canalizado
FR	:	Forma Revisada
FPI	:	Forma Produto da Inversa
CSR	:	*Compressed Sparse Row*
CSC	:	*Compressed Sparse Col*
MPS	:	*Mathematical Programming System Format*
LP	:	*Linear Programming File Format*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em diversas áreas surge a necessidade de organizar e usar os recursos disponíveis da maneira mais eficiente possível, recursos estes que podem ser tempo, matérias-primas, máquinas, operários, etc. Neste cenário surge uma área com suas vertentes após o final da Segunda Guerra Mundial [1], denominada Pesquisa Operacional (PO), sendo que neste período, diversas técnicas de aplicabilidade prática foram sistematizadas e formalizadas.

O maior expoente desta época foi o desenvonvimento e formalização do Método Simplex (MS) proposto por George B. Dantzig em 1947 [10]. Segundo Dongarra e Sullivan [13], o Algoritmo Simplex é considerado um dos dez mais influentes algoritmos da Ciência da Computação e Engenharia.

O MS se mostrou e ainda tem se apresentado como uma das principais ferramentas para solução não só de problemas práticos, mas também teóricos de Programação Linear (PL), Maros [34], problemas estes que consistem na minimização/maximização de um função linear sujeita a um conjunto de inequações/equações lineares, denominadas restrições. Devido a sua grande aplicabilidade, este método tem sido estudado e explorado, tanto do ponto de vista teórico quanto do ponto de vista computacional, sendo o segundo aspecto importantíssimo para que se tenha um implementações estáveis e que produzam soluções de maneira eficiente.

O uso de técnicas computacionais para implementações eficientes e estáveis de métodos tipo simplex são de extrema importância, Munari [25]. Os problemas reais de PL têm por característica serem altamente esparsos, Maros [43], e desta forma torna-se necessário que as estruturas de dados utilizadas na implementação tirem proveito desta característica, tanto em termos de armazenamento quanto em tempo computacional.

Cabe ressaltar que na grande maioria dos problemas reais de PL, o modelo ma-

temático é expressado de maneira canalizada nas restrições e variáveis. Como abordado por Sousa(2005) [40] este tipo de modelo é de particular interesse, visto que pode representar qualquer problema de PL. Assim, como também destacado por Sousa(2005) [40], este tipo de modelo pode ser especializado utilizando-se uma busca unidimensional.

Nesta dissertação, o método Dual Simplex para a classe de problemas lineares canalizados, tratado aqui como DSC, é apresentado e implementado utilizando três diferentes estruturas de dados e duas formas de atualização da base. A motivação para esta abordagem deve-se ao fato de se encontrar pouca referência na literatura dando ênfase a este tipo de implementação como apontado por Robert Bixby(2003) [5] e devido a sua facilidade de representação de modelos de problemas de PL.

Cabe ressaltar que uma das vantagens do método DSC abordado neste trabalho reside no fato de que o mesmo não necessita de uma fase inicial para obtenção de uma solução básica, e assim garante uma solução básica inicial factível, visto que pode ser iniciado a partir da base unitária.

Os experimentos computacionais foram aplicados usando 3 classes de problemas, sendo estas: 31 instâncias de problemas reais do repositório da NetLib[1], 15 instâncias de problemas reais oriundos da formulação de rações da fábrica de ração do Instituto Federal de Educação Ciência e Tecnologia do Espírito Santo - Campus de Alegre (IFES - Campus Alegre) e 36 instâncias geradas a partir de um gerador pseudo-aleatório de problemas de produção com alta esparsidade.

Em relação a atualização da base, foram implementadas as formas Revisada e Forma Produto da Inversa (FPI). Para cada uma das formas, 3 diferentes estruturas de dados foram utilizadas: estrutura densa ou indexada, conjuntos de vetores compactos (CSR/CSC) e listas encadeadas.

Um algoritmo foi desenvolvido para o processo de atualização da base na forma Revisada usando as estruturas CSR/CSC. Também foi dada particular atenção a estrutura da implementação sobre o ponto de vista de projeto de software e ao processo de entrada de dados, sendo que, para este último, foram desenvolvidos um conversor de arquivos do formato MPS para LP e um *parser* para leitura e organização dos dados em memória em estruturas de dados apropriadas. Esta etapa foi importantíssima, pois os conversores desenvolvidos podem ser úteis em outros projetos que necessitem fazer a leitura e *parser* de arquivos LP.

[1] `http://www.netlib.org/lp/data`

Por fim, em Bixby(2002) [4] reporta que o Método Dual Simplex, de um modo geral experimentalmente, possui uma performance superior ao Método Primal Simplex.

1.1 Objetivos

O trabalho tem como objetivo geral o estudo e implementação do método Dual Simplex especializado para a classe de problemas lineares canalizados utilizando diferentes estruturas de dados e formas de atualização da base.

Com as implementações pretende-se contribuir com a literatura de forma a possibilitar uma melhor tomada de decisão na escolha de estruturas de dados a serem consideradas na implementação do referido método, bem como as influências que tais estruturas provocam em seu comportamento levando em consideração as duas diferentes formas de atualização da base propostas.

Em síntese, como objetivos específicos do presente trabalho, espera-se:

- Realizar a implementação do método Dual Simplex especializado para a classe de problemas lineares canalizados.
- Utilizar duas formas de atualização da base:
  - Forma Revisada.
  - Forma Produto da Inversa.
- Para cada forma de atualização, utilizar três estruturas de dados distintas:
  - Forma Densa, ou Indexada.
  - Listas Encadeadas.
  - Conjunto de vetores compactos CSC/CSR.
- Efetuar testes com 3 classes de problemas:
  - Instâncias da fábrica de ração do IFES[2].
  - Instâncias altamente esparsas geradas de modo pseudo-aleatório.
  - Instâncias do repositório da NetLib[3].

[2]Instituto Federal do Espírito Santo - Campus de Alegre
[3]`http://www.netlib.org/`

1.2 Organização do Texto

No Capítulo 2 é realizada uma revisão bibliográfica com o intuito de introduzir os principais conceitos de Otimização Linear que constituem as bases do problema estudado. Já no Capítulo 3, o método Dual Simplex para problemas canalizados (DSC) é abordado, sua formulação matemática e sua estrutura são apresentadas e suas particularidades destacadas.

Em seguida, o Capítulo 4 trata sobre instabilidade numérica, pois métodos do tipo simplex possuem característica iterativa e sua implementação é realizada em uma máquina de precisão finita. Desta forma torna-se necessário a introdução de alguma técnica que evite a propagação de erros de arredondamento/truncamento. Neste capítulo também são abordadas as duas formas de atualização da base utilizadas no trabalho, forma Revisada e Forma Produto da Inversa (FPI). Ao final do capítulo, um algoritmo desenvolvido especialmente para atualização da base na implementação Revisada para as estruturas de vetores compactos CSC/CSR é apresentado.

O Capítulo 5 é dedicado às estruturas de dados utilizadas na implementação do método, os módulos desenvolvidos para tratamento da entrada de dados e organização do código sob o ponto de vista de projeto de software. Neste capítulo ainda é dada ênfase à esparsidade de problemas de Otimização Linear, bem como às estruturas de dados utilizadas para tornar mais eficiente as implementações que modelam tais problemas.

Por conseguinte, o Capítulo 6, os resultados computacionais da implementação são apresentados e discutidos, com ênfase na influência de cada estrutura de dados utilizada na solução de cada classe de problema abordado. Também para efeito de validação, os resultados obtidos também são comparados com a implementação do Dual Simplex presente no *solver* CPLEX 12.8. O Capítulo 7 é destinado à conclusão e propostas de trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo tem por finalidade realizar uma abordagem histórica da Programação Linear e por conseguinte da Pesquisa Operacional, destacando as principais contribuições para a consolidação desta importante área de planejamento estratégico. Também são enfatizados os principais temas atuais tratados na literatura relacionados a implementação de métodos do tipo simplex. Por final, é realizada a descrição e caracterização do problema de programação linear em sua versão primal e dual.

2.1 Programação Linear: Breve Histórico

A Pesquisa Operacional (PO) tem suas raízes com o surgimento do radar na Inglaterra, e por consequência, na busca de novas aplicações para este dispositivo, tais como interceptações de aeronaves inimigas. No início, o termo estava relacionado a criação de centros e seções da força aérea britânica destinados a aplicar técnicas a problemas de operações de guerra, tais como dimensionamento de frotas e manutenção de aviões. Como abordado por Arenales *et al.*(2011) [1], o uso sistemático de técnicas operacionais para racionalização de recursos militares teve seu ápice com a Segunda Gerra Mundial [11].

Após o término da Segunda Guerra Mundial, houve uma significativa evolução da PO na Inglaterra e nos Estados Unidos, culminando na criação de vários projetos governamentais e sociedades de pesquisas, tais como a *Operations Research Society of America - ORSA* nos Estados Unidos da América e *Operations Research Society - ORS* na Inglaterra [1]. Esta notável evolução se deu por conta do potencial das técnicas de pesquisa operacional poderem ser utilizadas nas mais diversas áreas, tais como indústria, agricultura, economia, transportes, etc [19].

Em 1947, o matemático norte americano George Bernard Dantzig trabalhava em um projeto que tinha por objetivo apoiar as operações da força aérea americana. Dantzig possuía na época uma extensa experiência na solução de problemas de planejamento de programação usando calculadoras de mesa. Então, foi desafiado por colegas a encontrar uma maneira de calcular mais rapidamente os processos de planejamento militar [11]. Neste período período, G. B. Dantzig desenvolveu, sistematizou e testou o algoritmo simplex em problemas de PL [10].

A PL, e sua generalização como ferramenta de extensa aplicabilidade prática, nas palavras do próprio Dantzig "era praticamente desconhecida antes de 1947" [11]. Em contrapartida, como destacado por ele, a evolução das técnicas de tomadas de decisão aconteceram de forma tão rápida, que não são muitos os que se lembram das principais contribuições anteriores daqueles que fundamentaram as bases para o que viria a se tornar a pesquisa operacional. Dentre estes, pode-se destacar John Von Neumann, Wassily Leontief , Leonid Kantorovich e Tjalling Koopmans. Kantorovich e Koopmans foram, em 1975, agraciados com o Nobel de Economia por suas contribuições à teoria de alocação ótima de recursos.

Em contraste com os economistas de seu tempo, como abordado por Bixby(2012) [6], Dantzig vislumbrou a PL não somente como uma ferramenta qualitativa para análise de fenômenos econômicos, mas como um método que poderia ser utilizado para computar as respostas às perguntas de problemas específicos do mundo real.

Uma das primeiras aplicações práticas reportadas do algoritmo simplex foi para a solução de uma instância do clássico problema da dieta de Stigler [41]. Este problema possuía um total de 21 restrições e 77 variáveis e o tempo total de computação necessário para sua solução era estimado em 120 homens/dias[1]. Em 1952 Dantzig e Orchard-Hays estavam trabalhando na *RAND Corporation* com o propósito de estabelecer o algoritmo simplex como um procedimento computacional bem definido. Vale ressaltar que naquela época os computadores eram bem diferentes dos que conhecemos atualmente e as primeiras implementações foram realizadas em máquinas que trabalhavam com tabulação de cartões e eram eletromecânicas. As implementações inicias não se mostraram muito promissoras, pois o cálculo da inversa da base era realizado de maneira explícita como descrito na forma algébrica, o que tornava o método computacionalmente inviável.

O algoritmo simplex é por natureza iterativo e trabalha com base no princípio combinatorial, então inicialmente existia o receio de que o algoritmo não fosse tão eficiente

[1]Unidade de medida com base em um número padrão de horas em um dia de trabalho

na prática e que sua complexidade fosse exponencial, Bazaraa(2010) [3]. Em 1972 Klee e Minty [28] demonstraram que para uma classe de problemas com $m = n$ restrições de igualdade, $2n$ variáveis não negativas e 2^n pontos extremos, o algoritmo simplex precisou realizar $2^n - 1$ iterações. Com este resultado ficou demonstrado que o algoritmo simplex não pertence a classe dos algoritmos polinomiais.

Apesar da demonstração de Klee e Minty em relação a complexidade do algoritmo simplex sobre o ponto de vista teórico, na prática, como destacado por Maros(2003) [34], o algoritmo simplex mostra um comportamento linear em função do número de restrições (m) e é altamente eficiente na solução de problemas reais. Esta característica polinomial apresentada na prática na solução de muitos problemas reais de otimização ficou conhecida como folclore do Algoritmo Simplex (Sousa(2000) [39] e Sousa(2005) [40]).

A busca por um algoritmo de complexidade polinomial para problemas de otimização linear começou então despertar interesse da comunidade científica e permaneceu em aberto até 1979, quando L. G. Khachiyan [27] propôs um algoritmo de complexidade polinomial, conhecido como Algoritmo dos Elipsóides. Este algoritmo por sua vez foi de grande importância teórica, mas se mostrou inviável na prática devido ao seu esforço para solucionar um problema estar estritamente ligado a magnitude do problema dado, tornando assim suas iterações altamente custosas.

Neste cenário, os métodos do tipo simplex continuaram sendo aperfeiçoados e implementados de maneira cada vez mais eficiente, não havendo concorrentes até 1984 quando Narendra Karmarkar [26] apresentou um novo algoritmo de complexidade polinomial usando geometria projetiva para solucionar problemas de programação linear. O algoritmo de Karmarkar foi o marco inicial para o que viria a ser conhecido como método de pontos interiores e potencializou as pesquisas na área de Programação Linear, tendo profunda influência para o avanço da Programação Linear Inteira (PLI). Bixby(2002)[4] destacou que apenas os métodos de pontos interiores apresentam desempenho competitivo em relação aos métodos tipo simplex.

Após o estabelecimento do algoritmo simplex como procedimento computacional viável para solução de problemas reais, várias foram as contribuições teóricas e práticas realizadas por inúmeros pesquisadores de diversas partes do mundo.

Com a expansão das capacidades de computação providas pelos computadores após a década de 70, pôde-se então, não somente resolver os problemas de forma mais rápida, mas também testar novas ideias e métodos que até então não seriam possíveis em máquinas de gerações anteriores [6], tais como o uso a fatoração LU para atualização da base

usando a abordagem de Forrest-Tomlim(1972) [15], sendo esta uma variante de Bartels-Golub [2] e melhoria no paradigma de seleção de variáveis, tal como proposto por Paula Harris(1973) [21].

Um dos primeiros saltos de desempenho computacional no algoritmo simplex foi a aplicação da técnica conhecida como Forma Produto da Inversa (FPI) proposta por Alex Orden [6]. A técnica consiste em calcular a inversa da base em uma determinada iteração realizando o produto de matrizes elementares, sendo que estas matrizes podem ser armazenadas de forma compacta, bastando um vetor e um indicador da coluna para cada matriz. Esta forma de se obter uma nova inversa traz vantagens em problemas de grande porte esparsos e provê uma melhor estabilidade numérica, pois evita que cálculos desnecessários sejam realizados.

Em 1954 Dantzig e Orchard-Hays [12] implementaram a técnica FPI no simplex e foi reportado que uma instância do problema da dieta de Stigler, como anteriormente citado, possuindo 26 restrições e 71 variáveis, pôde ser resolvido em um tempo total de 8 horas, sendo que boa parte deste tempo foi destinado a alimentar com cartões a máquina tabuladora da época [6].

O uso de técnicas da álgebra linear contribuíram de forma significativa para melhorar as implementações dos métodos tipo simplex. Entre estes desenvolvimentos destaca-se a fatoração LU aplicando a heurística de Markowitz para seleção de pivô, técnica esta que foi aprimorada por Suhl e Suhl [42] e é atualmente considerada mais eficiente e estável [4]; eficiente porque provoca um menor preenchimento no processo de atualização da base, conhecido como *fill-in*, e estável por proporcionar uma melhor estabilidade numérica durante as operações realizadas.

Para tirar proveito das técnicas citadas, as estruturas de dados utilizadas nas implementações de métodos tipo simplex devem ser escolhidas de maneira criteriosa, levando em conta como as informações são armazenadas, processadas e descartadas, como destacado por Munari [25]. Por exemplo, em problemas reais de grande porte, onde a esparsidade na matriz de restrições do problema é um fator predominante, deve-se então prezar por estruturas que não armazenem valores nulos, trazendo assim um ganho substancial nas rotinas de solução dos sistemas lineares presentes nas iterações do método simplex.

Tem-se dado particular atenção à técnicas para implementações eficientes de métodos do tipo simplex nos últimos tempos, incluido técnicas computacionais. No trabalho de Sousa(2005) [40] o método Dual Simplex para problemas canalizados em sua forma geral é abordado, onde é explorada sua característica de linearidade por partes e uma busca

unidimensional é proposta e testada. Também no mesmo trabalho, o método do Gradiente Bi-Conjungado é implementado e incorporado as rotinas do método simplex, onde o autor ressalta uma redução no tempo de solução de alguns problemas.

No trabalho de Koberstein(2005) [29], o autor aborda diversas técnicas para implementação do método Dual Simplex, tais como FPI e decomposição LU para atualização da base, exploração de hiper-esparsidade em algumas rotinas específicas, estabilidade numérica e tratamento de degeneração. Munari(2009) [25], assim como Koberstein, aborda no contexto do método primal simplex, diversas técnicas importantes para eficiência nas implementações e também apresenta algumas estruturas de dados que podem ser utilizadas para armazenamento de matrizes esparsas.

2.2 O Problema de Programação Linear

O problema de Programação Linear (PL) como encontrado na literatura está relacionado a maximização/minimização de uma função linear sujeito a um conjunto de restrições também lineares. As variáveis do problema devem atender a condições de não-negatividade. Assim sendo, podemos definir um problema de PL em notação matricial como:

$$Minimizar \;\; z(\mathbf{x}) = \mathbf{c}^{\intercal}\mathbf{x} \tag{2.1a}$$

$$Sujeito\ a \;\; A\mathbf{x} = \mathbf{b} \tag{2.1b}$$

$$\mathbf{x} \geq 0 \tag{2.1c}$$

em que A é uma matriz composta de m restrições (cada uma envolvendo um conjunto de variáveis) e n variáveis e os vetores $\mathbf{c}$ e $\mathbf{x} \in \mathbb{R}^n$ e o vetor $\mathbf{b} \in \mathbb{R}^m$. $\mathbf{x} \geq 0$ significa $x_j \geq 0 \; \forall \; j = 1, 2, \ldots, n$. A função $z(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ é chamada de função objetivo. $A\mathbf{x} = \mathbf{b}$ é chamado de conjunto de restrições do problema. Qualquer vetor $\mathbf{x}$ que satisfaça (2.1b) é chamado *solução*. Se esta solução também satisfaz (2.1c) então é denominada *solução factível*.

O conjunto de *soluções factíveis*, também denominado de *região factível* do problema (2.1), pode ser representado por:

$$X = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\} \tag{2.2}$$

Se $X = \varnothing$ diz-se que o problema é *infactível*. Resolver o problema (2.1), consiste em encontrar uma solução factível $\mathbf{x}^*$ tal que, $z(\mathbf{x}^*) \leq z(\mathbf{x})$ para toda *solução factível* $\mathbf{x}$, portanto diz-se que a solução $\mathbf{x}^*$ é ótima. Se $\forall\ \mathbf{x} \in X\ \exists\ \bar{\mathbf{x}} \in X \mid z(\bar{\mathbf{x}}) < z(\mathbf{x})$, então o problema é dito *ilimitado*, portanto não possui *solução ótima*.

As variáveis x_j, onde $j = 1, 2, \ldots, n$ são conhecidas como variáveis de decisão, nas quais seus valores são determinados de tal forma que as restrições em (2.1b) sejam satisfeitas e as condições em (2.1c) sejam atendidas.

O problema (2.1) encontra-se no formato denominado *forma padrão*. Com esta formulação todas as restrições estão representadas como igualdades e todas as variáveis estão sujeitas a não-negatividade. Nesta abordagem, implicitamente as *variáveis artificiais* de folga (quando existirem restrições com $\leq$) ou de excesso (quando existirem restrições com $\geq$) tratam estes casos de desigualdades.

Vale notar que quando se utiliza *variáveis artificiais* o problema original é ampliado, aumentando eventualmente o número de iterações e por conseguinte o tempo de solução.

O problema de PL também pode ser apresentado em um formato conhecido por *forma geral*, como mostrado abaixo em notação matricial:

$$Minimizar\ \ \bar{z}(\mathbf{x}) = c_0 + \bar{\mathbf{c}}^{\intercal}\mathbf{x} \tag{2.3a}$$

$$Sujeito\ a\ \ \mathbf{L} \leq \bar{A}\mathbf{x} \leq \mathbf{U} \tag{2.3b}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{2.3c}$$

em que $c_0 \in \mathbb{R}$, $\bar{\mathbf{c}}$, $\mathbf{x} \in \mathbb{R}^n$, $\bar{A} \in \mathbb{R}^{m \times n}$, os vetores $\mathbf{L}$ e $\mathbf{U} \in \mathbb{R}^m$, podendo também serem ilimitados, neste caso poderão assumir os valores $-\infty$ ou $+\infty$. Os vetores $\mathbf{L}$ e $\mathbf{U}$ são denominados, respectivamente, limitantes inferiores e superiores de cada restrição $\sum_{j=1}^{n} a_j^i x_j$ em que $i = 1, \ldots, m$. Os vetores $\mathbf{l}$ e $\mathbf{u} \in \mathbb{R}^n$, podem também assumir os valores $-\infty$ ou $+\infty$. Estes vetores são chamados de limitantes inferiores e superiores, respectivamente, de cada variável x_j, em que $j = 1, 2, \ldots, n$.

Uma determinada variável x_j em (2.3) pode ser caracterizada em uma das seguintes formas:

- *livre*: Se $l_j = -\infty$ e $u_j = +\infty$.

- *canalizada*: Se $l_j > -\infty$ e $u_j < +\infty$.
- *fixa*: Se $l_j = u_j = a$, onde $a \in \mathbb{R}$.

A constante aditiva c_0 em (2.3a) não altera o problema do ponto de vista das variáveis de decisão visto que, se x^* é uma solução ótima para (2.3a) menos a constante c_0, então $\mathbf{x}^*$ também é uma solução ótima para o problema (2.3a) com adição da constante c_0, em que $c_0 \in \mathbb{R}$; uma vez que $\bar{z}(\mathbf{x}^*) + c_0 \leq \bar{z}(\mathbf{x}) + c_0$.

O formato do problema (2.3) surge na modelagem de problemas reais, como, por exemplo, problemas da mistura na formulação de rações apresentados na Seção 6.1.1, problemas de programação da produção apresentados na Seção 6.1.2, assim como outros problemas da mistura para produção de ligas metálicas, tintas, adubos, etc.

De maneira geral, qualquer problema de PL pode ser manipulado e representado na *forma padrão* vista em (2.1), sendo este formato geralmente apresentado na literatura em textos cujo o objetivo é introduzir os conceitos iniciais de Otimização Linear, como pode ser visto em Maros(2003) [34].

A estrutura do problema (2.3) na *forma geral* com suas restrições e seus respectivos limitantes pode se vista na Figura 2.1 adaptada de Maros [34].

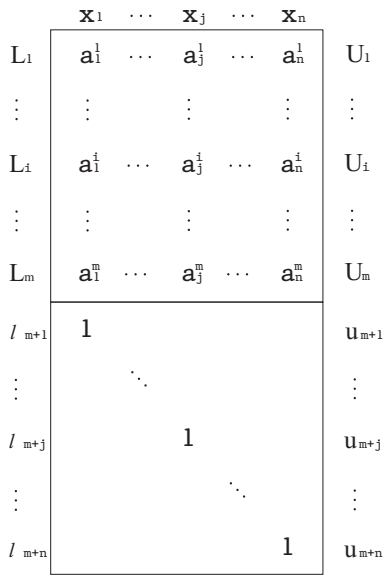

Figura 2.1: Estrutura das restrições e limitantes do problema de PL na forma geral.

O conjunto de *soluções factíveis* X definidas em (2.2) delimita o espaço de todas as *regiões factíveis* de um problema de PL. Este conjunto, geometricamente, pode ser

descrito como a interseção de um número finito de semiplanos, no qual é conhecido como *poliedro convexo*.

Os vértices do poliedro formado pelo conjunto em (2.2) correspondem as *soluções básicas factíveis*. Estes vértices são pontos extremos de X que não podem ser escritos como uma combinação linear não-trivial de outros pontos em X. A Figura 2.2 mostra um poliedro convexo no $\mathbb{R}^2$ tendo sua região factível em cinza e pontos extremos A, B, C, D e E.

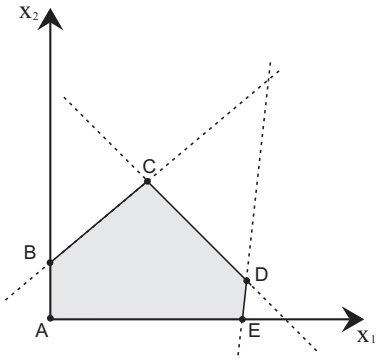

Figura 2.2: Representação de um poliedro convexo no $\mathbb{R}^2$.

2.2.1 O Problema Dual

O problema apresentado em (2.1) é conhecido como Primal. A cada problema Primal, existe um outro problema denominado Dual que pode ser representado como:

$$Minimizar \;\; z(\mathbf{y}) = \mathbf{b}^\intercal \mathbf{y} \tag{2.4a}$$

$$Sujeito\ a \;\; A^\intercal \mathbf{y} \leq \mathbf{c} \tag{2.4b}$$

$$\mathbf{y} \;\; irrestrito \tag{2.4c}$$

em que $A \in \mathbb{R}^{m \times n}$ e os vetores $\mathbf{y}$ e $\mathbf{b} \in \mathbb{R}^m$ e o vetor $\mathbf{c} \in \mathbb{R}^n$. A função (2.4a) é chamada de função objetivo dual. É importante notar no problema (2.4) que existe exatamente uma restrição dual para cada variável primal e exatamente uma variável dual para cada restrição primal encontrada em (2.1).

Pode-se então estabelecer algumas regras para a transformação de um problema *primal* em seu respectivo *dual* e vice-versa, como pode ser visto na Tabela 2.1.

Tabela 2.1: Regras de transformação Primal ↔ Dual.

	Primal	**Dual**	
Restrição	=	livre	Variável
	≤	≤	
	≥	≥	
Variável	≥	≤	Restrição
	≤	≥	
	livre	=	
	Vetor de recursos	Vetor de Custos	
	Vetor de Custos	Vetor de recursos	

Como o problema (2.4) é também um problema de PL, então este possui uma formulação dual associada, sendo que esta formulação consiste exatamente no problema (2.1). Assim sendo, o dual do dual fornece o modelo primal.

Existem duas outras propriedades importantes que relacionam os problemas primal/dual. Estas propriedades são formalizadas em dois teoremas conhecidos como Teorema da Dualidade Fraca (*Weak Duality*) e Teorema da Dualidade Forte (*Strong Duality*), como anunciados a seguir.

Teorema 1 (Dualidade Fraca)**.** *Seja uma solução primal factível arbitrária* $\mathbf{x}$ *para o problema* (2.1) *e qualquer solução dual* $\mathbf{y}$ *para o problema* (2.4)*, então os valores de suas funções objetivos estão sujeitos a condição de que* $\mathbf{b}^\intercal \boldsymbol{y} \leq \mathbf{c}^\intercal \mathbf{x}$.

Em linhas gerais, o Teorema (1) estabece que qualquer solução dual factível define um limitante inferior para a solução ótima da função objetivo primal. Do mesmo modo, qualquer solução primal factível estabelece um limitante superior para a solução ótima da respectiva função objetivo dual.

Teorema 2 (Dualidade Forte)**.** *Se o problema primal 2.1 possuir solução ótima* $\mathbf{x}^*$ *e o problema dual 2.4 também possuir uma solução ótima* $\mathbf{y}^*$*, então a igualdade* $\mathbf{x}^* = \mathbf{y}^*$ *se verifica.*

Detalhes relacionados as provas dos Teoremas (1) e (2) fogem ao escopo do presente trabalho e podem ser encontradas na literatura em textos que abordam de maneira mais abrangente as relações entre as formulações dos problemas primais e duais, tais como em Arenales(2011) [1], Bazaraa(2010) [3] e Maros(2003) [34].

Capítulo 3

Método Dual Simplex Para Problemas Canalizados

O método dual simplex foi proposto por Lemke em 1954 [32] e segundo Koberstein e Suhl [30] não se mostrava uma alternativa viável ao método primal simplex até o início dos anos 90. Como apontado em Bixby [6], inicialmente o método era utilizado quase que exclusivamente para lidar com reotimizações durante a solução de problemas inteiros mistos.

O cenário mudou com algumas contribuições, tais como as de Forrest e Goldfarb(1992) [14] e Fourer(1994) [16], onde Forrest e Goldfarb apresentam um estudo abrangente sobre a regra de Dantzig Normalizada (*steepest-edge rule*) fornecendo uma maneira menos custosa para o cálculo das normas das direções primal e Dual Simplex, como pode ser visto em Sousa(2005) [40]. No trabalho de Fourer é apresentado um teste da razão generalizado para o método Dual Simplex. Desde então, as implementações de *solvers* comerciais estabeleceram o método Dual Simplex como um algoritmo de propósito geral [6].

3.1 O Problema de PL com Restrições Canalizadas

O problema *Primal* de PL com restrições canalizadas pode ser representado da seguinte forma:

$$\begin{aligned} Minimizar \;\; & z(\mathbf{x}) = \mathbf{c}^{\intercal}\mathbf{x} \\ Sujeito\ a \;\; & \mathbf{l} \leq A\mathbf{x} \leq \mathbf{u} \end{aligned} \quad (\textbf{Primal}) \tag{3.1}$$

onde $A \in \mathbb{R}^{m+n \times n}, \mathbf{l}\ e\ \mathbf{u} \in \mathbb{R}^{m+n}$ com $l_i \leq u_i$, em que $i = 1, \ldots, m+n$.

A matriz A de (3.1) pode ser particionada de acordo com (3.2).

$$A = \left[\begin{array}{c} A' \\ \hline I \end{array}\right] \tag{3.2}$$

em que $A' \in \mathbb{R}^{m \times n}$ formado pelas restrições técnicas do problema primal e $I \in \mathbb{R}^{(m+1)\times(m+n)}$ formando assim uma matriz identidade composta pela canalização das variáveis.

Pode-se supor, sem perda de generalidade que o posto da matriz A é definido pelas suas colunas, ou seja, igual a n. Para desenvolver o problema dual do problema (3.1) deve-se transformá-lo em um problema linear na forma padrão sem ampliar o número de restrições, caso contrário, acarretaria em um considerável esforço computacional. Para isso, pode-se definir $\mathbf{y} = A\mathbf{x}$, então o problema (3.3) é equivalente ao problema (3.1):

$$\begin{array}{c} Minimizar \;\; z(\mathbf{x}) = \mathbf{c}^\intercal \mathbf{x} \\ Sujeito\ a \;\; A\mathbf{x} - \mathbf{y} = 0 \\ \mathbf{l} \leq \mathbf{y} \leq \mathbf{u} \end{array} \tag{3.3}$$

Observe que o problema (3.1) é a forma mais geral, pois como destacado em (3.2), existe uma submatriz identidade em A. Assim as restrições de não-negatividade (ou canalizações) nas variáveis, que são bastante comuns em alguns exemplos práticos, aparecem explicitamente. Se ainda $l_i = u_i$ onde $i = m+1, \ldots, m+n$, serão produzidas restrições de igualdade.

Pode-se determinar o problema dual de (3.1) usando o problema equivalente (3.3). Para isto define-se a função Lagrangiana:

Seja $\boldsymbol{\lambda} \in \mathbb{R}^m$,

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) &= \mathbf{c}^\intercal \mathbf{x} + \boldsymbol{\lambda}^\intercal (\mathbf{y} - A\mathbf{x}) \\ &= (\mathbf{c}^\intercal - \boldsymbol{\lambda}^\intercal A)\mathbf{x} \; + \; \boldsymbol{\lambda}^\intercal \mathbf{y}, \boldsymbol{\lambda} \in \mathbb{R}^n \end{aligned} \tag{3.4}$$

Considere o problema Lagrangiano:

Para cada $\boldsymbol{\lambda} \in \mathbb{R}^m$, seja:

$$h(\boldsymbol{\lambda}) = \{min_{(x,y)}\ L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) \mid \mathbf{l} \leq y \leq \mathbf{u},\ \text{onde}\, i = 1, \ldots, n\} \tag{3.5}$$

Da definição em (3.5), segue-se a seguinte desigualdade: $z(\mathbf{x}) \geq h(\boldsymbol{\lambda})\ \forall\ \mathbf{x}$ factível ($\mathbf{y} - A\mathbf{x} = 0$, em que $L(\boldsymbol{x}, \mathbf{y}, \boldsymbol{\lambda}) = \mathbf{c}^\intercal \mathbf{x}$) e $\forall\ \boldsymbol{\lambda} \in \mathbb{R}^m$. Ou seja, $h(\boldsymbol{\lambda})$ é um limitante inferior para $z(\mathbf{x})$. Esta desigualdade é a motivação para definição do chamado *problema dual*, em que consiste na determinação do melhor limitante inferior, ver Sousa(2005) [40].

Como $\mathbf{x}$ em (3.5) é irrestrito de sinal e $\mathbf{y}$ é canalizado, restringi-se $\boldsymbol{\lambda}$ tal que o mínimo em (3.5) exista. Assim, tem-se:

$$\mathbf{c}^\intercal - \boldsymbol{\lambda}^\intercal A = 0 \ \text{ ou } \ A^\intercal \boldsymbol{\lambda} = \mathbf{c} \tag{3.6}$$

Portanto, pode-se definir o problema dual como:

$$\begin{aligned} &Maximizar\ \ h(\boldsymbol{\lambda}) \\ &Sujeito\ a: A^\intercal \boldsymbol{\lambda} = \mathbf{c} \end{aligned} \tag{3.7}$$

Vale notar que em (3.7) as variáveis de folga estão implicitamente no vetor $\boldsymbol{\lambda}$. Busca-se agora uma expressão para a função objetivo dual $h(\lambda)$. Usando a restrição (3.6) em (3.4) tem-se:

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}, \lambda) &= \lambda^\intercal \mathbf{y} \\ &= \sum_{i=1}^{m} \lambda_i y_i \end{aligned} \tag{3.8}$$

Então,

$$\begin{aligned} h(\boldsymbol{\lambda}) = L(\mathbf{x}, \mathbf{y}, \boldsymbol{\lambda}) &= \sum_{i=1}^{m} \lambda_i l_i + \sum_{i=1}^{m} \lambda_i u_i \\ l_i \leq y_i \leq u_i \quad i/\lambda_i > 0 \quad &i/\lambda_i < 0 \end{aligned} \tag{3.9}$$

se $\lambda_i = 0$, y_i pode assumir qualquer valor no intervalo $[l_i, u_i]$. Assim é possível

explicitar a função objetivo dual como:

$$h(\boldsymbol{\lambda}) = \sum_{i=1}^{m} h_i(\lambda_i) \tag{3.10}$$

onde

$$h_i(\lambda_i) = \begin{cases} \mathbf{l}_i\lambda_i & se\ \lambda_i < 0 \\ \\ \mathbf{u}_i\lambda_i & se\ \lambda_i > 0 \end{cases} \qquad i = \{1, \ldots, m\}$$

Ou seja, o problema dual é um problema de programação linear onde a função objetivo é linear por partes e a Figura 3.1 representa o significado desta função.

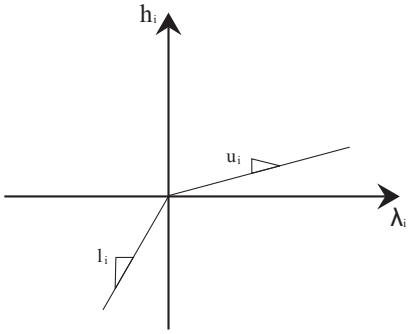

Figura 3.1: Função linear por partes.

Desta maneira pode-se obter finalmente o problema dual:

$$\begin{aligned} &Maximizar\ \ L(\boldsymbol{\lambda}) = \sum_{i=1}^{m} h_i(\lambda_i) \\ &Sujeito\ a\ \ A^{\intercal}\boldsymbol{\lambda} = \mathbf{c} \end{aligned} \qquad (\textbf{Dual}) \tag{3.11}$$

Note que o problema dual abordado sempre será factível, pois o posto $(A) = n$ e $\boldsymbol{\lambda}$ livre. Se este for finito (por conseguinte, o primal factível), haverá uma solução ótima básica: $A^{\intercal} = (B^{\intercal}, N^{\intercal})$ onde $B^{\intercal}$ é a matriz formada pelas colunas (linhas do primal) básicas e $N^{\intercal}$ a matriz formada pelas não básicas.

Observe que a solução geral do sistema de restrições do problema (3.6) é dada por:

$$A^{\intercal}\boldsymbol{\lambda} = \mathbf{c} \Leftrightarrow \mathbf{B}^{\intercal}\boldsymbol{\lambda}_B + \mathbf{N}^{\intercal}\boldsymbol{\lambda}_N = \mathbf{c} \Leftrightarrow \boldsymbol{\lambda}_B = (\mathbf{B}^{\intercal})^{-1}\mathbf{c} - (\mathbf{B}^{\intercal})^{-1}\mathbf{N}^{\intercal}\boldsymbol{\lambda}_N \tag{3.12}$$

O interesse está apenas em maximizar $L(\boldsymbol{\lambda})$, onde $\boldsymbol{\lambda}$ pode assumir qualquer valor. Assim, em particular, é possível fazer:

$$\hat{\boldsymbol{\lambda}} = \begin{bmatrix} \hat{\boldsymbol{\lambda}}_B \\ \hat{\boldsymbol{\lambda}}_N \end{bmatrix} \text{ com } \hat{\boldsymbol{\lambda}}_B^{\intercal} = \mathbf{c}^{\intercal} B^{-1} \text{ e } \hat{\boldsymbol{\lambda}}_N^{\intercal} = 0 \quad \text{(solução básica dual)} \tag{3.13}$$

Considere agora, as restrições do problema primal particionadas por linhas:

$$\begin{bmatrix} B \\ N \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{bmatrix} \Rightarrow \begin{cases} B\mathbf{x} = \mathbf{y}_B \\ N\mathbf{x} = \mathbf{y}_N \end{cases} \tag{3.14}$$

Da primeira equação em (3.14) vem que:

$$\hat{\mathbf{x}} = B^{-1}\mathbf{y}_B, \quad \text{(Solução básica primal)} \tag{3.15}$$

substituindo (3.15) na segunda equação de (3.14):

$$\hat{\mathbf{y}}_N = NB^{-1}\mathbf{y}_B \tag{3.16}$$

Os valores de $\mathbf{y}_B$ em (3.16) são determinados por:

$$y_{Bi} = \begin{cases} l_i, & se\ (c^{\intercal}B^{-1})_i \geq 0 \\ u_i, & se\ (c^{\intercal}B^{-1})_i < 0 \end{cases} \qquad i = \{1, \ldots, m\} \tag{3.17}$$

Observando que a restrição $l_N \leq y_N \leq u_N$ foi relaxada e y_B é definido por (3.17). Desta forma, como resultado, dada a partição $A = \begin{bmatrix} B \\ N \end{bmatrix}$ e as soluções associadas (3.13) e (3.15). Se as restrições $\mathbf{l}_N \leq \mathbf{y}_N \leq \mathbf{u}_N$ forem satisfeitas, então as soluções primal e dual corrente são as soluções ótimas. Para mais detalhes do método Dual Simplex ver Sousa(2005) [40] e Koberstein(2005) [29].

Observe que, se for aplicado o método primal ao problema (3.1), a principal dificuldade é obter uma solução factível inicial caso seja disponível. O método das variáveis artificiais leva a uma ampliação do problema, enquanto que o método de remoção de infactibilidade exige a implementação de um novo algoritmo.

A estratégia *dual simplex* consiste na perturbação de apenas uma variável dual não básica, ou seja, apenas uma componente de $\boldsymbol{\lambda}_N$. Assim, existem dois casos a serem

considerados (**Caso (a)** e **Caso (b)**). Para simplificação, supõe-se que a solução dual é não degenerada, caso contrário, $\lambda_{B_i} = 0$ para algum $i \in \mathbf{B}$.

Caso (a) : A variável λ_{N_q} é acrescida da seguinte forma:

$$\begin{cases} \lambda_{Nq} = \delta, & \text{com } \delta \geq 0 \text{ e pequeno} \\ \lambda_{Ni} = 0, & i = 1, \ldots, m-n \text{ e } i \neq q. \end{cases}$$

Ou seja:

$$\boldsymbol{\lambda}_N = \delta \hat{\mathbf{e}}_q, \tag{3.18}$$

em que $\hat{e}_q$ é a q-ésima coluna da matriz identidade $(m-n) \times (m-n)$.

Com esta perturbação nas variáveis não básicas, determina-se a perturbação nas variáveis básicas de tal forma que o sistema $\mathbf{A}^\top \lambda = \mathbf{c}$ seja satisfeito. Para isto, substituiu-se (3.18) em (3.12) e será obtido:

$$\lambda_B = (\mathbf{B}^\top)^{-1}\mathbf{c} - \delta(\mathbf{B}^\top)^{-1}\mathbf{a}^\intercal{}_{Nq}$$

De (3.13), tem-se que:

$$\boldsymbol{\lambda}_B = \hat{\lambda} + \delta\eta_B,$$

Sendo $\boldsymbol{\eta}_B = -(\mathbf{B}^\top)^{-1}\mathbf{a}^\intercal{}_{Nq}$ o vetor das componentes básicas da direção Dual Simplex. Então, a nova solução dual pode ser escrita da seguinte forma:

$$\lambda = \begin{pmatrix} \lambda_B \\ \lambda_N \end{pmatrix} = \begin{pmatrix} \hat{\lambda}_B \\ \hat{\lambda}_N \end{pmatrix} + \delta \begin{pmatrix} -(\mathbf{B}^\top)^{-1}\mathbf{a}^\intercal{}_{Nq} \\ \hat{e}_q \end{pmatrix},$$

ou ainda,

$$\lambda = \hat{\lambda} + \delta\eta, \tag{3.19}$$

onde

$$\eta = \begin{pmatrix} -(\mathbf{B}^\top)^{-1}\mathbf{a}^\intercal{}_{Nq} \\ \hat{e}_q \end{pmatrix} \tag{3.20}$$

é chamada de *direção dual simplex*. A parte superior de (3.20) se refere a partição básica e a inferior a não básica.

Como as variáveis duais são irrestritas de sinal, o crescimento de δ não precisa ser limitado pela função objetivo dual $h(\boldsymbol{\lambda})$. Isto sugere uma busca na direção $\boldsymbol{\eta}$ para a

escolha de δ. A busca é linear por partes devido a característica da função objetivo dual. Esta estratégia não foi considerada neste trabalho, mas como foi demonstrado por Sousa(2005) [40], faz reduzir muito a quantidade de iterações.

A perturbação da solução básica dual em (3.19) com a direção dada promove uma variação da função objetivo dual que é dada por:

$$h(\boldsymbol{\lambda}) = h(\hat{\boldsymbol{\lambda}}) + \delta(-\mathbf{a}^{\intercal}{}_{Nq}\hat{\mathbf{x}} + d_{Nq}) \tag{3.21}$$

onde d_{Nq} corresponde ao coeficiente na função dual de λ_{Nq} Para mais detalhes deste desenvolvimento, consulte Sousa [40].

Portanto, se $d_{Nq} - \mathbf{a}^{\intercal}{}_{Nq}\hat{\mathbf{x}} > 0$, ou seja, se o limite inferior da q-ésima restrição primal for violado, então a *direção dual simplex* é uma direção de subida.

A expressão (3.21) é válida, desde que λ_B em (3.19) não sofra mudança de sinal, pois caso contrário o coeficiente da função objetivo dual é alterado. Se $\hat{\lambda}_{B_i}$ e η_{B_i} têm sinais opostos, então λ_{B_i} pode trocar de sinal com o crescimento de δ. Portanto, dois casos devem ser considerados:

i) $\hat{\lambda}_{B_i} < 0$ e $\eta_{B_i} > 0$, então impondo que $\hat{\lambda}_{B_i} + \delta\eta_{B_i} \leq 0$, segue que: $\delta \leq -\frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}$;

ii) $\hat{\lambda}_{B_i} < 0$ e $\eta_{B_i} < 0$, então impondo que $\hat{\lambda}_{B_i} + \delta\eta_{B_i} \geq 0$, segue que: $\delta \leq -\frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}$.

Desta forma, para que não haja mudança de sinal em $\hat{\lambda}_{B_i}$ deve-se ter:
$\delta \leq \delta_i = \frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}$, $\forall\, i = 1, \ldots, n$ tal que $\hat{\lambda}_{B_i}$ e η_{B_i} tenham sinais opostos.

Seja p um índice básico, tal que:

$$\delta_p = -\frac{\lambda_{B_p}}{\eta_{B_p}} = min\left\{-\frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}, \text{tal que } -\frac{\hat{\lambda}_{B_i}}{\eta_{B_i}} \geq 0, i = 1, \ldots, n\right\} \tag{3.22}$$

Para $0 \leq \delta \leq \delta_p$ temos que: $h(\lambda) = h(\hat{\lambda}) + \delta(d_{N_q} - \hat{\mathbf{y}}_{N_q})$. Quando $\delta = \delta_p$ tem-se, de (3.19) e (3.22), que p-ésima restrição básica torna-se nula: $\lambda_{B_p} = 0$, o que sugere uma redefinição da partição básica, onde λ_{B_p} torna-se não básica e $\lambda_{N_q} = \delta_p > 0$ torna-se básica.

Caso (b) : A variável λ_{Nq} é decrescida da seguinte forma:

$$\begin{cases} \lambda_{Nq} = -\delta, \ \text{ com } \delta \geq 0 \text{ e pequeno} \\ \lambda_{Ni} = 0, \ \ i = (1, \ldots, m-n) \text{ e } i \neq q. \end{cases}$$

Ou seja:

$$\lambda_N = -\delta \hat{\mathbf{e}}_q \tag{3.23}$$

O procedimento é análogo ao **Caso (a)**, mas vale notar agora que:

$$\lambda_B = \hat{\lambda}_B - \delta \eta_B$$

em que $\boldsymbol{\eta}_B = -(\mathbf{B}^{\intercal})^{-1}\mathbf{a}^{\intercal}{}_{N_q}$ é o vetor das componentes da *direção dual simplex*. Portanto a nova solução dual pode ser escrita como:

$$\boldsymbol{\lambda} = \begin{pmatrix} \lambda_B \\ \lambda_N \end{pmatrix} = \begin{pmatrix} \hat{\lambda}_B \\ \hat{\lambda}_N \end{pmatrix} - \delta \begin{pmatrix} -(\mathbf{B}^{\top})^{-1}\mathbf{a}^{\intercal}{}_{Nq} \\ \hat{e}_q \end{pmatrix}$$

ou ainda,

$$\boldsymbol{\lambda} = \hat{\boldsymbol{\lambda}} - \delta\eta, \tag{3.24}$$

Fazendo o mesmo desenvolvimento da função objetivo dual para a nova solução (3.24) e considerando as devidas mudanças, segue que:

$$h(\boldsymbol{\lambda}) = h(\hat{\boldsymbol{\lambda}}) + \delta(\mathbf{a}^{\intercal}{}_{N_q}\hat{\mathbf{x}} - \mathbf{e}_{N_q}) \tag{3.25}$$

Portanto, se $\mathbf{a}^{\intercal}{}_{N_q}\hat{\mathbf{x}} - \mathbf{e}_{N_q} > 0$, ou seja, se o limite superior da q-ésima restrição primal for violado, então a *direção dual simplex* é uma direção de subida.

A expressão (3.25) é válida, desde que λ_B não sofra mundança de sinal. Logo, se $\hat{\lambda}_B$ e η_B têm o mesmo sinal, então λ_B pode trocar de sinal com o crescimento de δ. Portanto, de maneira análoga ao **Caso (a)**, para que não haja mudança de sinal em λ_{B_i} , devemos ter:

$$\delta < \delta_i = \frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}$$

Seja p um índice básico, tal que:

$$\delta_p = \frac{\lambda_{B_p}}{\eta_{B_p}} = min\left\{\frac{\hat{\lambda}_{B_i}}{\eta_{B_i}}, \text{tal que } \frac{\hat{\lambda}_{B_i}}{\eta_{B_i}} < 0, i = (1, \ldots, n)\right\} \tag{3.26}$$

Para $0 \leq \delta \leq \delta_p$ temos que: $h(\lambda) = h(\hat{\lambda}) + \delta(\hat{\mathbf{y}}_{N_q} - \mathbf{e}_{N_q})$. Quando $\delta = \delta_p$ tem-se que p-ésima restrição básica torna-se nula, enquanto que $\lambda_{N_q} = -\delta \leq 0$, o que sugere uma redefinição da partição básica realizando a troca dos índices e repetindo o procedimento.

A seguir, é apresentado o Algoritmo DSC para o método Dual Simplex canalizado sem considerar a aplicação da busca unidimensional.

3.1.1 Algoritmo Dual Simpex Canalizado

Tomando por base o problema dual apresentado em (3.11) e suas particularidades, pode-se apresentar o algoritmo DSC que descreve o método para problemas na forma canalizada. O Algoritmo 1 é apresentado em *pseudocódigo*, sendo assim, detalhes de implementação computacional são omitidos, visto que o objetivo é demonstrar de maneira conceitual seus principais passos.

O Algoritmo 1 começa com a base dual inicial *factível* de inicialização imediata, formada pelas colunas unitárias da partição na matriz A, sendo esta partição composta pela canalização das variáveis. Iterativamente o método segue alternando entre bases enquanto a função dual puder ser melhorada e a solução ótima ser encontrada, ou o problema se mostrar dual ilimitado.

Algoritmo 1 *Dual Simplex Canalizado - DSC*

Entrada: Matriz A, vetores $\mathbf{b}, \mathbf{l}$ e $\mathbf{u}$; Base dual inicial factível I.

Saída: Solução ótima $\mathbf{x}$ ou problema dual ilimitado.

P0: **Inicialização** (Inicialmente $B = I \therefore B^{-1} = I$)

Defina os conjuntos dos índices-básicos (I_B) e não-básicos (I_N):

$I_B = \{m+1, m+2, \ldots, m+n\}$ e $I_N = \{1, 2, \ldots, m\}$

Calcule a solução dual. Inicialmente $\hat{\lambda}_B = \mathbf{c}$.

Calcule a solução primal $\hat{x} = B^{-1}\hat{y}_B$ onde $\hat{y}_B$ é definido como:

$$\hat{y}_{B_i} = \begin{cases} \mathbf{l}_{B_i} & \text{se } \hat{\lambda}_{B_i} \geq 0 \\ \mathbf{u}_{B_i} & \text{se } \hat{\lambda}_{B_i} < 0 \end{cases} \quad \text{onde } i = 1, \ldots, n$$

continue = VERDADEIRO

enquanto (continue) **faça**

P1: **Teste de otimalidade**

Calcule $\hat{y}_N = (N\hat{\mathbf{x}})_{m\times 1}$ e verifique a maior violação para $\mathbf{l}_N \leq \hat{\mathbf{y}}_N \leq \mathbf{u}_N$.

se ($\mathbf{l}_{N_j} \leq \hat{\mathbf{y}}_N \leq \mathbf{u}_{N_j} \ \forall\, j \in I_N$) **então**

continue = FALSO

//solução ótima encontrada

senão

Seja $q \in I_N$ que determina a maior violação primal, seja no limite inferior ou superior. O índice q entrará na base (B).

fim se

P2: **Direção Dual Simplex**

Calcule $\eta_B = (B^\top)^{-1}(a_{q.})^\top$, em que $a_{q.}$ representa a q-ésima linha da partição não básica da matriz A.

P3: **Tamanho do passo:**

se (houver violação no limite superior) **então**

Calcule

$$\delta_i = \left\{ \ -\left(\frac{\hat{\lambda}_B}{\eta_{B_i}}\right) \mid -\left(\frac{\hat{\lambda}_B}{\eta_{B_i}}\right) \geq 0, \ \ i = 1, \ldots, n \ \right\}$$

senão

Calcule

$$\delta_i = \left\{ \ \frac{\hat{\lambda}_B}{\eta_{B_i}} \mid \frac{\hat{\lambda}_B}{\eta_{B_i}} \geq 0, \ \ i = 1, \ldots, n \ \right\}$$

Ordene os deltas (δ_i) onde $i = 1, \ldots, r$ (r deltas).

Selecione $\delta_p = min\{\delta_i, \ i = 1, \ldots, r\}$, sendo que $p \in I_B$. O índice p será o índice que sairá da base.

fim se

P4: **Atualização**

- Atualize o conjunto de índices básicos (I_B) e o conjunto de índices não básicos (I_N), desta forma, o I_{B_p} se torna o q-ésimo índice básico e I_{N_q} se torna o p-ésimo índice não-básico.

- Atualize a solução dual ($\overline{\lambda}$):
  $\overline{\lambda}_B = \lambda_B + \delta\eta_B$

- Atualize a nova inversa da base ($\overline{B}^{-1}$).

- Atualize a solução primal ($\overline{x}$):
  $\overline{x} = \overline{B}^{-1}\overline{y}_B$

fim enquanto

Para ilustrar o funcionamento do Algoritmo 1, o exemplo apresentado em (3.27), representando um problema primal no espaço $\mathbb{R}^2$ será utilizado.

$$
\begin{array}{llrcrcrl}
\textbf{Minimizar} & & z(\mathbf{x}) & = & -1\mathrm{x}_1 & -2\mathrm{x}_2 & & \\
\textbf{Sujeito a} & \mathsf{R1} & 0 & \leq & 1\mathrm{x}_1 & -1\mathrm{x}_2 & \leq & 2 \\
 & \mathsf{R2} & -4 & \leq & 2\mathrm{x}_1 & +1\mathrm{x}_2 & \leq & 4 \\
 & \mathsf{R3} & -1 & \leq & 1\mathrm{x}_1 & & \leq & 2 \\
 & \mathsf{R4} & -1 & \leq & & 1\mathrm{x}_2 & \leq & 2
\end{array}
\tag{3.27}
$$

A Figura 3.2 mostra o polígono convexo que compõe a região primal factível formada pelas restrições do problema (3.27). Também encontram-se destacados na figura os pontos extremos do polígono.

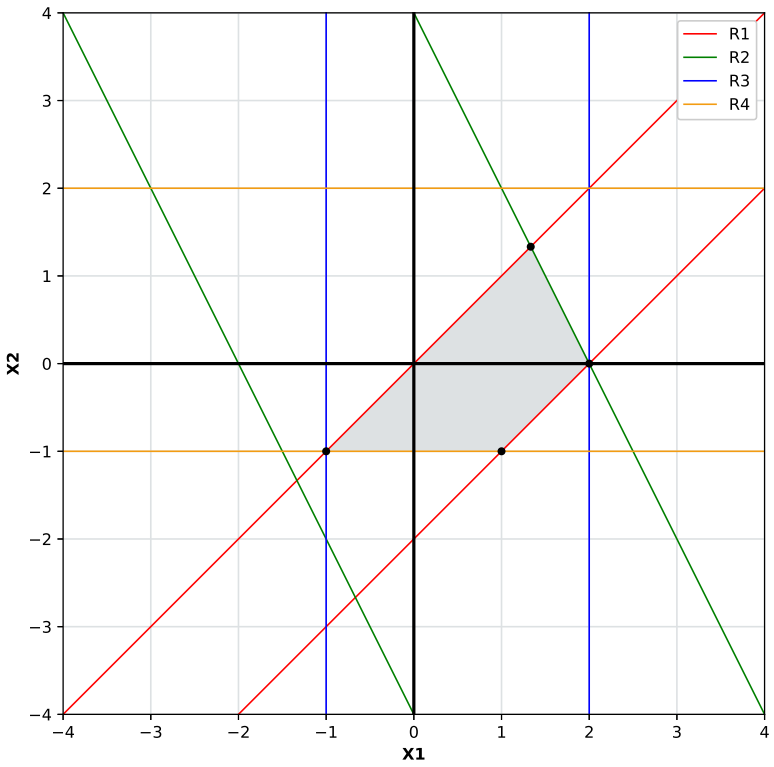

Figura 3.2: Região primal factível formada pelas restrições R1, R2, R3 e R4.

O problema dual correspondente ao problema primal (3.27) é apresentado em (3.28).

$$\begin{aligned} &\textbf{Maximizar } \; L(\boldsymbol{\lambda}) = \sum_{i=1}^{4} h_i(\lambda_i) \\ &\textbf{Sujeito a } \begin{cases} 1\lambda_1 + 2\lambda_2 + 1\lambda_3 &= -1 \\ -1\lambda_1 + 1\lambda_2 + 1\lambda_4 &= -2 \end{cases} \end{aligned} \tag{3.28}$$

Onde,

$$h_1(\lambda_1) = \begin{cases} 0 & \text{se } \lambda_1 > 0 \\ 2\lambda_1 & \text{se } \lambda_1 < 0 \end{cases}$$

$$h_2(\lambda_2) = \begin{cases} -4\lambda_2 & \text{se } \lambda_2 > 0 \\ 4\lambda_2 & \text{se } \lambda_2 < 0 \end{cases}$$

$$h_3(\lambda_3) = \begin{cases} -1\lambda_3 & \text{se } \lambda_3 > 0 \\ 2\lambda_3 & \text{se } \lambda_3 < 0 \end{cases}$$

$$h_4(\lambda_4) = \begin{cases} -1\lambda_4 & \text{se } \lambda_4 > 0 \\ 2\lambda_4 & \text{se } \lambda_4 < 0 \end{cases}$$

Para aplicação do Algoritmo 1, inicialmente deve-se particionar a matriz A, assim:

$$A = \left(\frac{N}{B}\right) = \left(\begin{array}{cc} 1 & -1 \\ 2 & 1 \\ \hline 1 & 0 \\ 0 & 1 \end{array}\right)$$

Com o particionamento de A, pode-se definir os conjuntos dos índices básicos (I_B) e índices não básicos (I_N):

$$\begin{aligned} I_B = \{3,4\} &\Rightarrow I_{B_1} = 3 \;\; e \;\; I_{B_2} = 4 \\ I_N = \{1,2\} &\Rightarrow I_{N_1} = 1 \;\; e \;\; I_{N_2} = 2 \end{aligned}$$

Iteração 1:

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad N = \begin{pmatrix} 1 & -1 \\ 2 & 1 \end{pmatrix}$$

- Cálculo da solução dual:
  $\hat{\lambda}_B = (B^\top)^{-1}\mathbf{c}$, mas como inicialmente $(B^\top)^{-1} = I$, então $\hat{\lambda}_B = \mathbf{c}$. Desta forma:

$$\hat{\lambda}_B = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$$

- Cálculo da solução básica primal associada:
  $\hat{y}_{B_1} = \hat{y}_3 = 2$
  $\hat{y}_{B_2} = \hat{y}_3 = 2$
  $B\hat{x} = \hat{y}_B$
  $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$
  $\hat{x} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$
  $\hat{y}_N = N\hat{x} = \begin{pmatrix} 1 & -1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \end{pmatrix}$

  Na Figura 3.3 pode ser notado que o método DSC parte, inicialmente, de uma solução primal infactível $\mathbf{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$.

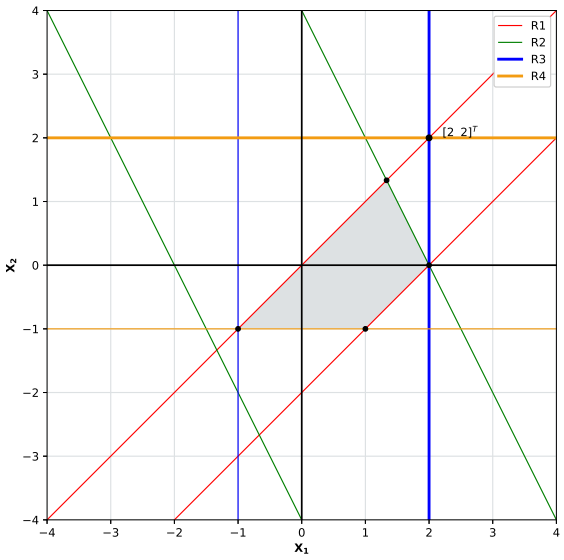

Figura 3.3: Solução inicial primal infactível - Restrições R3 e R4 ativas.

Também encontram-se em destaque na Figura 3.3, as restrições ativas neste ponto de execução do DSC. Como pode ser visto, as restrições 3 e 4 ($I_B = \{3, 4\}$), encontram-se ativas no seu limitante superior ($\mathbf{u}_B$).

- Realização do teste de otimalidade:
  $N_1 = 1 : \hat{y}_{N_1} = 0$ (Restrição não violada)
  $N_2 = 2 : \hat{y}_{N_2} = 6$ (Restrição violada no limite superior)

  A solução atual não é ótima, pois a condição de canalização referente a restrição R_2 não foi satisfeita.

  - Determinando o índice q a entrar na base:
    A restrição violada foi N_2, portanto: $q = 2$, pois $\hat{y}_{N_2} > u_{N_2}$.

- Cálculo da direção dual simplex:
  $\eta_B = (B^\top)^{-1}(a_{N_2})^\top$

$$\eta_B = \begin{pmatrix} (B^\top)^{-1}(a_{N_2})^\top \\ e_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

- Cálculo do tamanho do passo:

  - Determinando os valores dos deltas (δ) :
    $\delta_1 = -(-\frac{1}{2}) = \frac{1}{2}$
    $\delta_2 = -(-\frac{2}{2}) = 1$

  - Ordenando os valores dos deltas:
    $\delta_1 = \frac{1}{2}$
    $\delta_2 = 1$

  - Selecionando $\delta_p = min\{\delta_i,\ \ i = 1, \ldots, r\}$, então $\delta_1 = \frac{1}{2}$ sairá da base. Neste caso $p = 1 \mid p \in I_B$ sairá da base.

- Atualização:

  - Atualização do conjunto de I_B e I_N:
    $I_{B_1} \leftrightarrow I_{N_2}$, portanto:

$$I_B = \{2, 4\} \quad \Rightarrow I_{B_1} = 2 \ \ e \ \ I_{B_2} = 4$$
$$I_N = \{1, 3\} \quad \Rightarrow I_{N_1} = 1 \ \ e \ \ I_{N_2} = 3$$

$$B = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \qquad N = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix}$$

– Atualização da solução dual ($\overline{\lambda}_B$):
$\overline{\lambda}_B = \lambda_B + \delta\eta_B$
$\overline{\lambda}_B = \begin{pmatrix} -1 \\ -2 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} \\ -\frac{3}{2} \end{pmatrix}$

– Cálculo da nova inversa da base ($\overline{B}^{-1}$):

$$B = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \Rightarrow \overline{B}^{-1} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{pmatrix}$$

– Atualização da solução primal associada ($\overline{x}$):
$\overline{x} = \overline{B}^{-1}\overline{y}_B$
$\overline{x} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

A Figura 3.4 mostra que após a atualização da solução primal, o método DSC moveu para uma nova solução primal infactível $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

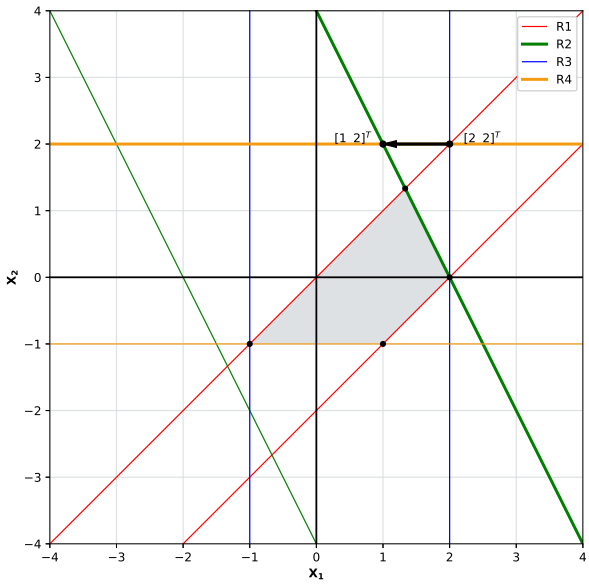

Figura 3.4: Nova solução primal infactível - Restrições R2 e R4 ativas.

Como pode ser observado na Figura 3.4, as restrições ativas no limitante superior neste ponto da execução do DSC são 2 e 4 ($I_B = \{2, 4\}$).

Neste etapa, uma nova iteração do método DSC seria realizada. Com esta nova iteração, o método moveria para a solução ótima primal factível do problema 3.27, que neste caso é $\hat{\mathbf{x}}^* = \begin{bmatrix} 1.\overline{3} \\ 1.\overline{3} \end{bmatrix}$. A solução ótima pode ser vista na Figura 3.5, representada em forma de dízima periódica, juntamente com o gradiente da função objetivo (∇z).

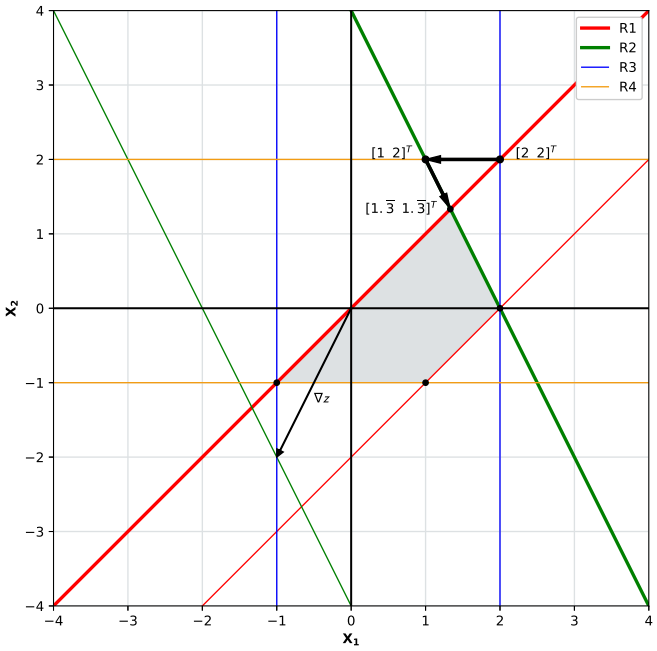

Figura 3.5: Solução ótima do problema 3.27 - Restrições R1 e R2 ativas.

Cabe destacar que o método Dual Simplex trabalha com soluções primais infactíveis e obtém a factibilidade primal (solução ótima) quando esta existe, ou o problema se tornar dual ilimitado.

Capítulo 4

Estabilidade Numérica e Formas de Atualização da Base

4.1 Estabilidade Numérica

Em uma arquitetura de precisão finita, como é o caso dos computadores digitais, os cálculos realizados por aplicações que necessitam de maior precisão podem ser afetados. Desta forma, em métodos do tipo simplex, que são por natureza iterativos, erros de arredondamento, truncamento e cancelamento podem ocorrer fazendo com que a solução de um determinado problema não seja encontrada ou, se encontrada, não corresponda a solução correta. Assim sendo, torna-se necessário um tratamento numérico adequado para minimizar estas instabilidades.

4.1.1 Degeneração e tolerâncias numéricas

Durante as iterações em métodos simplex pode ocorrer que o valor de uma das variáveis básicas se torne nulo, desta forma o valor da função objetivo poderá não se alterar, fazendo com que a solução corrente não possa ser melhorada. Geometricamente a degeneração pode ser vista como um ponto extremo onde há interseção de mais do que n hiperplanos, sendo n o número de variáveis de um problema. O número de hiperplanos maiores que n é conhecido como ordem de degeneração, Bazaraa (2010) [3].

Na ausência de degeneração, e assumindo a factibilidade de um determinado problema, o método simplex irá convergir em um número de iterações finitas para uma solução factível ótima ou ilimitada. Caso haja degeneração, poderá ocorrer que após um número finito de passos, o método retorne a uma base idêntica a de um anterior, formando assim

um ciclo de bases. Este fenômeno é conhecido na literatura como ciclagem.

O não tratamento de degeneração poderá fazer com o que método fique em um *loop* infinito fazendo com que não haja convergência. Sousa (2005) [40] *apud* Maros (2003) [34] destaca que a existência da degeneração poderá vir a provocar ciclagem, mas na prática dificilmente isso ocorre.

Como citado por Munari (2009) [25], existem algumas técnicas disponíveis na literatura para o tratamento de degeneração e ciclagem, como a regra de Bland e a regra lexicográfica, garantindo a convergência teórica do método. Entretanto, em implementações computacionais, como também destacado pelo autor, estas regras não são eficientes e na prática são adotadas algumas heurísticas tais como perturbações e expansão de limitantes, Gill *et al.* (1989) [18] e Bixby (2002) [4].

Para lidar com a estabilidade numérica de métodos tipo simplex algumas tolerâncias devem ser consideradas. Em cada iteração, diversas comparações e operações aritméticas são realizadas utilizando valores de ponto flutuante, desta forma como já citado, em uma arquitetura de precisão finita estes valores podem acumular erros de arrendamento, truncamento e anulamentos podem ser realizados provocando assim instabilidade numérica.

Segundo Koberstein (2005) [29], para garantir certa estabilidade, tolerâncias devem ser introduzidas no código, tais como tolerância que evite a escolha de um pivô muito pequeno no teste da razão, evitando assim possíveis passos degenerados. Também são consideradas tolerâncias para lidar com valores muito próximos de zero e cancelamento de elementos menores que um determinado valor estabelecido.

Na implementação do DSC há algumas perturbações, sendo estas representadas por meio de tolerâncias. Estes valores estão descritos na Tabela 4.1.

Tabela 4.1: Tolerâncias utilizadas.

Tolerância	**Sigla**	**Valor**
Comparação com zero	`TOL_COMP_ZERO`	$10^{-12} - 10^{-8}$
Degeneração	`TOL_DEGEN`	10^{-8}
Violação de limitantes	`TOL_VIO_LIM_INF_SUP`	10^{-8}

Os valores da torelância `TOL_COMP_ZERO` utilizados, como pode ser visto na tabela anterior, variaram de 10^{-12} a 10^{-8}. Estes valores foram escolhidos após vários testes e verificou-se que para as classes de problemas testados estas seriam boas escolhas. As tolerâncias `TOL_COMP_ZERO` e `TOL_DEGEN` são utilizadas na rotina do cálculo do tamanho do passo destacado no Algoritmo 1 do Capítulo 3. Esta rotina retorna a variável candidata a

sair da base atual, sendo que a escolha desta variável se dá pela seleção do menor valor de δ_i obtido. O valor de δ_i é calculado por meio da razão $\hat{\lambda}_B/\eta_{B_i}$, desta forma `TOL_COMP_ZERO` é utilizado para verificar se o valor desta razão é muito próximo de zero, caso isto aconteça poderá então ocorrer um passo degenerado.

A tolerância `TOL_VIO_LIM_INF_SUP` é utilizada na rotina que realiza o teste de otimalidade descrita no Algoritmo 1 do Capítulo 3. O valor da tolerância é utilizado com o objetivo de provocar uma relaxação nos limitantes inferiores e superiores ao se determinar a maior violação primal. Após a execução desta rotina, o índice que provoca a maior violação primal será escolhido para entrar e compor a nova base. Esta tolerância também é importante pois pode permitir que com esta ligeira relaxação, o método possa conseguir escapar de passos nulos, evitando assim também a degeneração.

Existem outras técnicas para tratamento de degeneração como a técnica *EXPAND* em Gil et al. [18] e teste de Harris [21]. Estas não foram utilizadas neste trabalho, mas podem ser objetos de estudos futuros.

4.2 Procedimentos de Atualização da Base

A cada iteração do método simplex é necessário realizar a atualização da representação da matriz inversa da base. Executar este processo de maneira explícita como descrita na álgebra torna-se computacionalmente custoso, assim é necessário encontrar maneiras mais eficientes. Existem algumas formas de atualização que se mostram mais adequadas quando implementadas em meio computacional, tais como a forma revisada, forma produto da inversa e atualização por meio da fatoração LU. Neste trabalho foram implementadas as formas revisada e produto da inversa, estas formas serão abordadas nas próximas seções.

4.2.1 Forma Revisada

Nesta forma de atualização, suponha a matriz B^{-1}, de dimensão n, como sendo a matriz inversa da base na iteração t, onde $t \neq 1$, pois para $t = 1$ tem-se $B^{-1} = I$, pois por construção o método DSC sempre é possível obter $B^{-1} = I$, o que caracteriza uma

grande vantagem do método. Esta matriz pode ser representada da seguinte forma:

$$B^{-1} = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\boldsymbol{p},1} & b_{\boldsymbol{p},2} & \dots & b_{\boldsymbol{p},n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{bmatrix} \tag{4.1}$$

O valor p representa o índice da linha que sairá da base na iteração corrente. A nova inversa da base, denotada por $\overline{\mathrm{B}}^{-1}$ pode ser obtida realizando-se a multiplicação de uma matriz de transformação elementar E pela matriz (4.1). Sendo que esta matriz de transformação difere da matriz identidade somente pela k-ésima coluna, onde k refere-se a posição no conjunto de índices básicos (I_B) associada ao índice p básico. Desta forma a matriz de transformação E é dada por:

$$E = \begin{bmatrix} 1 & 0 & \dots & \frac{-\eta_{b_1}}{\eta_{b_k}} & \dots & 0 \\ 0 & 1 & \dots & \frac{-\eta_{b_2}}{\eta_{b_k}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\eta_{b_k}} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{-\eta_{b_n}}{\eta_{b_k}} & \dots & 1 \end{bmatrix} \tag{4.2}$$

Onde, a coluna k é definida em função das componentes do vetor da direção Dual Simplex (η_B) na qual é obtida fazendo-se $\eta_B = (B^\top)^{-1}(a_{q.})^\top$. Em que $a_{q.}$ representa a q-ésima linha da partição não básica da matriz A, sendo q o índice que entrará base ($q \in I_N$ - De acordo com o Algoritmo 1) e determina a maior violação primal, seja no limite inferior ou superior. Vale destacar que a matriz (4.2) pode ser armazenada computacionalmente armazenando apenas a coluna k e seu respectivo valor.

Vale notar na matriz (4.2), que para a linha $i = k$, a componente será calculada fazendo $\frac{1}{\eta_{b_k}}$ e para as demais componentes será obtida por $\frac{-\eta_{b_i}}{\eta_{b_k}}$.

A nova matriz inversa da base, é obtida então pelo produto das matrizes (4.1) e (4.2),

como pode ser visto na equação (4.3).

$$\overline{B}^{-1} = EB^{-1} \tag{4.3}$$

Na inicialização do DSC tem-se que $B = I$ (matriz base inicial), logo de acordo com a igualdade de (4.3) que a base é $\overline{B}^{-1} = E$.

O processo de atualziação das demais linhas de $\overline{B}^{-1}$ (quando $i \neq k$) é obtido realizando a adição da i-ésima linha de B^{-1} com a k-ésima linha de B^{-1} multiplicada por $\frac{-\eta_{b_i}}{\eta_{b_k}}$, ou seja:

$$\overline{B}_{i,k}^{-1} = \begin{cases} b_{i.} + (\frac{-\eta_{b_n}}{\eta_{b_k}})b_{k.}, & se\ i \neq k \\ -\frac{1}{\eta_{b_k}}b_{k.}, & se\ i = k \end{cases} \qquad i = 1, \ldots, n \tag{4.4}$$

Após a realização do processo de atualização da base acima descrito, então será necessário atualizar a solução primal e continuar o processamento descrito no Algoritmo 1.

4.2.1.1 Complexidade das Operações no Cálculo da Inversa da Base

Tendo em vista que a operação de maior custo no DSC consiste na atualização da base, então com o objetivo de mensurá-la foi realizado o cálculo de complexidade para a forma revisada.

Neste caso, para cada linha da nova inversa da base ($\overline{B}^{-1}$), com ($i \neq k$) e $\eta_{b_i} \neq 0$, no cenário de pior caso, devem ser consideradas as seguintes operações:

- 1 divisão $\frac{-\eta_{b_i}}{\eta_{b_k}}$
- n multiplicações $\frac{-\eta_{b_i}}{\eta_{b_k}}b_{k.}$, onde $b_{k.} = [b_{k1}\ \ b_{k2} \ldots b_{kn}]$
- n subtrações $\left(b_{i1} - \frac{\eta_{b_i}}{\eta_{b_k}}b_{k1}\right)\left(b_{i2} - \frac{\eta_{b_i}}{\eta_{b_k}}b_{k2}\right)\ldots\left(b_{im} - \frac{\eta_{b_i}}{\eta_{b_k}}b_{km}\right)$

Totalizando $2n + 1$ operações. Estas operações serão realizadas por um total de $n - 1$ linhas, então:

Total parcial de Operações $= (n - 1)(2n + 1)$

Ainda tem-se as operações da linha $i = k$:

- n divisões $\frac{-1}{\eta_{b_k}}b_{k.}$, onde $b_{k.} = [b_{k1}\ \ b_{k2} \ldots b_{kn}]$

Tendo assim um aditivo de n operações, ou seja:
Total parcial de Operações $= (n-1)(2n+1) + n = 2n^2 - 1$.

Portanto, para cada linha $(i \neq k)$ com $\eta_{b_i} = 0$ haverá $(2n+1)$ operações a menos. Logo o total de operações serão:

T $= (2n^2 - 1) - (2n+1)nl$, em que nl corresponde ao número de linhas que atendem a condição citada.

4.2.2 Forma Produto da Inversa

Como já mencionado no Capítulo 2, uma dos primeiros avanços em implementações eficientes de métodos do tipo simplex, foi a FPI proposta em 1954 por Dantzig e Orchard-Hays [12] para atualização da inversa da base em cada iteração. A partir deste ponto, constatou-se que utilizar técnicas adequadas para o armazenamento e operações com a inversa da base são essenciais nas implementações computacionais do método simplex.

Como abordado na seção anterior, a nova matriz inversa da base $(\overline{B}^{\top})^{-1}$ pode ser obtida pela multiplicação da matriz básica atual $(B^{\top})^{-1}$ com a matriz E. Esta matriz difere da matriz identidade somente por uma única componente, representada pela coluna k.

Devido a particular estrutura da matriz E, esta pode ser armazenada em um único vetor, denominado neste caso em particular vetor $\mathsf{eta}(\boldsymbol{\eta})$, juntamente com o valor da coluna k associado ao índice p básico. Assim, a matriz pode ser representada pelo par $(\boldsymbol{\eta}, k)$. A ideia de armazenar o conjunto de matrizes elementares em um conjunto de vetores eta é extendida de forma que a matriz inversa da base na iteração t é dada pelo produto dos vetores eta das $t-1$ iterações.

Inicialmente tem-se que a matriz inversa inicial é dada $(B^{\top})^{-1} = I$ de ordem n. Na t-ésima iteração, a matriz inversa é dada por:

$$(B^{\top})^{-1}{}_t = E_t E_{t-1} \dots E_2 E_1 \tag{4.5}$$

A maneira de obter a matriz inversa da base na iteração t de acordo com a Equação (4.5) é conhecida como Forma Produto da Inversa. Este fato permite atualizar a matriz inversa da base sem inverter a base atual de forma explícita a cada iteração. Então, a solução de

um sistema na iteração t no método simplex pode ser obtida da seguinte maneira:

$$\begin{aligned} (B^\top)_t z &= v \\ (B^\top)^{-1}{}_t (B^\top)_t z &= (B^\top)^{-1}{}_t v \\ Iz &= (B^\top)^{-1}{}_t v \\ z &= (E_t E_{t-1} \dots E_2 E_1) v \end{aligned} \tag{4.6}$$

Se $\alpha = E_1 v$, então temos a seguinte representação matricial para solução do sistema:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \vdots \\ \alpha_{n-1} \\ \alpha_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & \boldsymbol{\eta}_1 & \cdots & 0 \\ 0 & 1 & \cdots & \boldsymbol{\eta}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \boldsymbol{\eta_k} & \vdots & \vdots \\ 0 & 0 & \cdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \boldsymbol{\eta}_n & \cdots & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}$$

O produto acima é dado por:

$$\alpha_i = \begin{cases} v_i + \boldsymbol{\eta}_i v_k & i = 1, \dots, n \text{ onde } i \neq k \\ \boldsymbol{\eta}_k v_k & i = k \end{cases}$$

Vale notar que quando $v_k = 0$, então $\alpha = v$. Esta condição deve ser verificada antes de realizar os cálculos acima para evitar que operações desnecessárias sejam executadas. Desta forma, $E_2\alpha$ pode ser calculado usando o resultado $\alpha = E_1 v$, de modo que a solução do sistema (4.6) poderá ser obtida fazendo $t-1$ chamadas ao procedimento acima. Como as matrizes elementares do sistema (4.6) são utilizadas em ordem crescente de índice, então esta transformação é conhecida como FTRAN (*Forward Transformation*).

A operação acima, também pode ser descrita em um sistema onde as componentes α e v são transpostas, ou seja $\alpha^\top = v^\top E$. Neste caso a solução será dada por:

$$\alpha_i = \begin{cases} v_i & i = 1, \dots, n \text{ onde } i \neq k \\ \displaystyle\sum_{j=1}^{n} \eta_j v_j & i = k \end{cases}$$

No caso acima, as matrizes de transformação elementares serão utilizadas de ma-

neira decrescente em relação seus índices, então dizemos que esta operação é denominada BTRAN (*Backward Transformation*).

Ambas transformações (FTRAN e BTRAN) são essenciais para solução de sistemas lineares nas iterações do método simplex. Nas implementações realizadas no presente trabalho com a FPI, o cálculo da direção Dual Simplex, que é obtida fazendo $\eta_B = (B^\top)^{-1}(a_{q.})^\top$, em que $a_{q.}$ representa a q-ésima linha da partição não básica da matriz A variando de $1..n$, é feito com a transformação FTRAN, visto que $(B^\top)^{-1}$ encontra-se armazenada como um conjunto de vetores eta($\boldsymbol{\eta}$). Já a transformação BTRAN é utilizada no cálculo da solução primal, sendo obtida por meio da expressão $x = B^{-1}y_B$ como descrito no Algoritmo 1.

Os passos representando os procedimentos em *pseudocódigo* para o cálculo da direção Dual Simplex com FTRAN e solução primal usando BTRAN encontram-se sumarizados nos Algoritmos 2 e 3 respectivamente.

Algoritmo 2 *Algoritmo para o cálculo da direção dual simplex usando FTRAN na FPI*

Entrada: Índice a entrar na base q, conjunto de matrizes elementares (`mtzs_elem`), matriz A (`mtz_a`), dimensão da matriz B (`dim_mtx_b`).
Saída: Vetor com as componentes da direção dual (`vdir_dual`) calculadas.

```
1: - Faça o vdir_dual ← mtz_a[q][.]
2: // para cada matriz de transformação elementar, realize FTRAN
3: para  i = 1 até num_mtzs_elem passo 1 faça
4:    k ← mtzs_elem[i].k;
5:    aux ← vdir_dual[k]
6:    // verificação para não realizar operações desnecessárias
7:    se aux > 0 então
8:       // percorra o vetor eta_i (η_i) e atualize vdir_dual para j ≠ k
9:       para j = 1 até dim_mtx_b passo 1 faça
10:         se j ≠ k então
11:            vdir_dual[j] ← vdir_dual[j] + aux * mtzs_elem[i].vet_eta[j];
12:         fim se
13:      fim para
14:      // atualize vdir_dual na posição k
15:      vdir_dual[k] ← aux * mtzs_elem[i].vet_eta[k];
16:   fim se
17:   aux ← 0
18: fim para
```

Algoritmo 3 *Algoritmo para o cálculo da solução primal usando BTRAN na FPI*

Entrada: Conjunto de matrizes elementares (`mtzs_elem`), dimensão da matriz B (`dim_mtx_b`), vetor Y_b (`vet_yb`) e solução primal atual x (`vet_sol_primal`).
Saída: Vetor com as componentes da solução primal atualizadas

```
1: soma ← 0
2: - Faça o vet_aux ← vet_yb
3: // Pós-multiplique vet_yb por cada matriz de transformação elementar
4: para  i = num_mtzs_elem até 1 passo -1 faça
5:    k ← mtzs_elem[i].k;
6:    // percorra o vetor eta_i (η_i)
7:    para j = 1 até dim_mtx_b passo 1 faça
8:       soma ← soma + mtzs_elem[i].vet_eta[j] * vet_aux[j]
9:    fim para
10:   vet_aux[k] ← soma
11:   soma ← 0
12: fim para
13: - Faça vet_sol_primal ← vet_aux
```

É importante destacar que todas as operações nos algoritmos anteriores apresentados estão sendo realizadas levando-se em conta a estrutura indexada (densa) e não as estruturas de Listas Encadeadas e conjunto de vetores compactos CSC/CSR.

4.2.2.1 Complexidade das Operações no Cálculo da Solução Primal Utilizando a FPI

Considere o cálculo para a obtenção da solução primal como sendo $x = B^{-1}\hat{y}_B$. De acordo com o Algoritmo 3, na implementação do presente trabalho a solução primal é obtida usando a transformação BTRAN, ou seja, $x^\top = \hat{y}_B^\top B^{-1}$.

Levando em consideração o armazenamento das matrizes elementares em conjunto de vetores compactos CSC como mostrado na Figura 5.5 e as 3 principais operações descritas na linha 8 do Algoritmo 3, então pode-se supor que o número de elementos não-nulos (nnz) na representação de cada vetor eta é sempre menor ou igual a dimensão da base (n), ou seja, $\mathsf{nnz}_t \leq n$. Desta forma, na primeira iteração tem-se $3nnz_1$ operações, a partir da segunda iteração irá incorrer que $x^\top = \hat{y}_B^\top E_2 E_1$, então serão $(3nnz_2)+(3nnz_1) = 3(nnz_2 + nnz_1)$. Na t-ésima iteração, o número de operações será:

$$T(n) = 3(nnz_t + nnz_{t-1}, \ldots, nnz_2 + nnz_1]$$

Com base na complexidade anterior, suponha uma matriz básica de dimensão nxn e

20% de não zeros para cada matriz elementar $E^{\top}$. Em t iterações tem-se:

$$T(n) = 3\left[\left(\frac{n}{5}\right)_t + \left(\frac{n}{5}\right)_{t-1}, \ldots, \left(\frac{n}{5}\right)_2 + \left(\frac{n}{5}\right)_1\right]$$
$$T(n) = 3t\frac{n}{5}$$

Empiricamente, como abordado em Sousa(2005) [40], o número de iterações do método simplex é uma função polinomial de grau baixo em relação ao número de linhas do problema. Sob esta ótica, suponhamos o número de iterações igual a 1,5n e $n = 1000$, deste modo:

$$T(n) = 3t\frac{n}{5}$$
$$T(n) = \frac{3\text{x}1,5n^2}{5}$$
$$T(n) = 900000 \text{ operações}$$

Para efeito de comparação, a complexidade para o DSC com a implementação na forma Revisada Indexada no cálculo da solução primal em cada iteração é: $T(n) = 3n^2$. Assim, tomando por base os mesmos dados do exemplo anterior para o número de iterações e dimensão da base, tem-se:

$$T(n) = 3tn^2$$
$$T(n) = 3000000\text{x}1,5n$$
$$T(n) = 4,5\text{x}10^9 \text{ operações}$$

Como pode ser visto, o número de operações é muito maior para a forma Revisada, o que justifica em parte o excelente desempenho da FPI com vetores CSC/CSR frente a forma revisada indexada no Capítulo 6.

Capítulo 5

Estruturas de Dados, Esparsidade e Implementação Computacional

Neste capítulo são discutidas as estruturas de dados utilizadas no desenvolvimento e ressaltadas algumas de suas características. Também é dada particular atenção a um assunto diretamente associado a problemas reais de PL, a esparsidade. Por fim, alguns detalhes da estruturação e implementação do código, bem como o processo de entrada de dados são apresentados.

5.1 Estruturas de Dados

Um dos principais fatores associados a implementação computacional de métodos do tipo simplex está na escolha do tipo de estrutura de dados a ser utilizada. Segundo Maros [34] não há uma estrutura de dados ideal que atende a todos os propósitos de uma maneira geral. Desta forma elas devem ser escolhidas levando em conta as operações realizas durante a execução do método.

Existem basicamente dois tipos de estrutura de dados a serem consideradas na implementação de um método computacional: as estruturas estáticas e dinâmicas. Nas estruturas estáticas, a quantidade de memória necessária a ser reservada e utilizada é conhecida *a priori*, e esta não muda no decorrer da execução do algoritmo. Em contrapartida, nas estruturas classificadas como dinâmicas, não se conhece, a princípio, a quantidade de memória necessária para armazenamento dos dados, sendo necessário gerenciá-la durante toda a execução.

Uma estrutura comumente tratada como estática nas implementações de métodos do tipo simplex é a responsável por armazenar a matriz A de restrições do problema,

pois esta permanece imutável durante todo o processamento do algoritmo. As estruturas dinâmicas, geralmente são necessárias durante o processo de inversão/fatoração da matriz básica, onde a princípio não se conhece o número de novos elementos que poderão surgir nesta etapa.

Como destacado por Munari(2009) [25], além das representações das matrizes e vetores, estruturas de dados que sejam eficientes também são necessárias para armazenar os mais diversos tipos de informação de um problema de PL internamente, tais como o conjunto de índices das variáveis básicas e não-básicas, soluções primais e duais, limitantes inferiores, superiores, etc.

5.2 Esparsidade

A esparsidade é uma das principais características que permitem que problemas de programação linear sejam possíveis de serem resolvidos utilizando computadores, como destacado por Maros [34]. Esta característica está diretamente relacionada a problemas reais de PL, ela pode ser definida em função do número de elementos não-nulos presentes na matriz de coeficientes do problema com dimensões $m \times n$, como já abordado anteriormente.

O conceito de esparsidade também é visto como baixa densidade, uma vez que a densidade da matriz A de um problema de PL pode ser calculada pela razão entre o número de elementos não-nulos e o total de elementos presentes na matriz. Deste modo, seja nnz_a o número de elementos não-nulos em A e ρ a densidade, então $\rho(A)$ pode ser calculada conforme a Equação (5.1):

$$\rho(A) = \frac{nnz_a}{mn} \tag{5.1}$$

Durante o processamento realizado nas iterações de métodos do tipo simplex, elementos que inicialmente eram nulos podem vir a se tornar não-nulos, fenômeno conhecido na literatura como *fill-in* ou preenchimento. Deste modo, é importante a preservação da esparsidade durante os cálculos, mas segundo Julian [20], preservá-la constitui-se um dos maiores desafios da álgebra computacional.

Os elementos de uma matriz de um problema de PL são armazenados no meio computacional usando precisão dupla, geralmente por um *double* de 8 *bytes*. Desta forma, como

exemplificado por Maros [34], o armazenamento explícito da matriz A de um problema de PL iria requerer $8mn$ *bytes*. Suponha uma matriz com 10000 linhas e 20000 colunas, isso totalizaria 200 milhões de elementos, necessitando, portanto de quase $1,5$ *gigabytes* para serem armazenados. Neste caso, não é somente o número de elementos que importa, mas também o número de operações realizadas em cada iteração do método com tais elementos, o que inviabiliza este tipo de tratamento para problemas de grande porte na prática.

Com o objetivo de tirar proveito da esparsidade presente nos problemas e otimizar as computações em cada iteração do método, estruturas de dados específicas são utilizadas com este propósito, tais como conjunto de vetores esparsos e listas encadeadas.

5.2.1 Vetores Esparsos

Antes de abordar as estruturas utilizadas no presente trabalho que tiram proveito da esparsidade, é dada uma visão geral sobre algumas formas de armazenamento de vetores em meio computacional.

A princípio, como destacado por Koberstein [29], existem três abordagens para armazenar vetores no meio computacional, sendo estas:

- `Densa`: em que todos os elementos, incluindo nulos, são armazenados de maneira explícita.
- `Indexada`: faz o uso de um *array* secundário de forma conjunta a abordagem densa. Este *array* tem por finalidade guardar os índices dos elementos não-nulos presentes na forma densa.
- `Compacta`: nesta forma, somente são armazenados os valores não-nulos e seus respectivos índices. Uma variável também pode ser utilizada para armazenar o tamanho do *array*.

A abordagem densa, na prática, não se mostra muito viável sobre o ponto de vista de armazenamento, mas possui certas vantagens, como o acesso simples e imediato, seu tamanho é conhecido *a priori* e a realização de operações são de fácil implementação [34]. Porém, isto tem um preço, visto que operações desnecessárias com elementos nulos são realizadas.

Na abordagem indexada existe a vantagem das operações poderem ser realizas somente com os elementos não-nulos. Em contrapartida, estes ainda continuam sendo armazenados

de maneira desnecessária. Por fim a abordagem compacta se mostra a mais viável em termos de espaço e tempo para realização das operações.

Suponha o vetor **v** em (5.2).

$$\mathbf{v} = \begin{bmatrix} 2.5 & 0.0 & 7.1 & 3.0 & 0.0 & 0.0 & 9.6 \end{bmatrix} \tag{5.2}$$

Sua representação nas três formas citadas podem ser vistas na Figura 5.1.

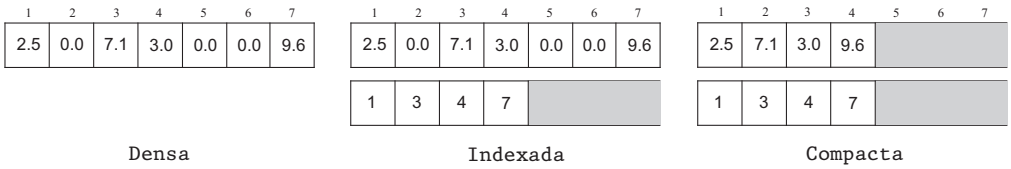

Figura 5.1: Representações do vetor **v** nas formas `densa`, `indexada` e `compacta`.

As componentes do vetor **v**, como podem ser vistas, encontram-se ordenadas em sua representação `compacta`, contudo não há obrigatoriedade quanto a manutenção dos elementos em uma ordem pré-definida. A parte em cor cinza nas representações `indexada` e `compacta` representam espaços livres, caso seja necessário a inserção de novos elementos.

As operações envolvendo dois vetores armazenados de forma compacta podem ser realizadas fazendo com que um dos vetores seja expandido, em seguida a operação é realizada e posteriormente o vetor é restaurado em sua forma compacta. Este procedimento pode, a princípio, parecer pouco viável, mas na prática o número de operações a serem executadas está em função dos elementos não-nulos e independem do tamanho original dos vetores participantes [34].

Para o armazenamento e realização de operações que tiram proveito da esparsidade em matrizes e vetores, existem duas abordagens formadas por conjunto de vetores compactos. Estas estruturas são conhecidas como *Compressed Sparse Row (CSR)* ou Estrutura Compacta por Linhas e *Compressed Sparse Column (CSC)* também denominada Estrutura Compacta por Colunas.

5.2.1.1 Estrutura Compacta por Linhas (CSR)

Nesta forma de armazenamento, os elementos não nulos da matriz são organizados em conjunto de três vetores, sendo estes denominados `val`, `ptr_lin` e `ind_col`, onde `val` armazena os valores não-nulos (nnz) na ordem em que estes aparecem nas linhas da matriz, `ptr_lin` possui referências para as posições em `val` nas quais uma nova linha da matriz se inicia e `ind_col` guarda o valor das colunas nas quais cada elemento em `val` está associado. Como exemplo, segue a matriz quadrada $M_{m\times m}$, com $m = 4$ mostrada em (5.3).

$$M = \begin{bmatrix} 4.0 & 0.0 & 0.0 & -2.0 \\ 0.0 & 1.9 & 0.0 & 0.0 \\ 0.0 & 0.0 & -5.0 & 7.5 \\ 6.0 & 3.0 & 0.0 & 8.7 \end{bmatrix} \tag{5.3}$$

Os vetores `val`, `ind_col` e `ptr_lin` para a matriz M (5.3) são mostrados na Figura 5.2.

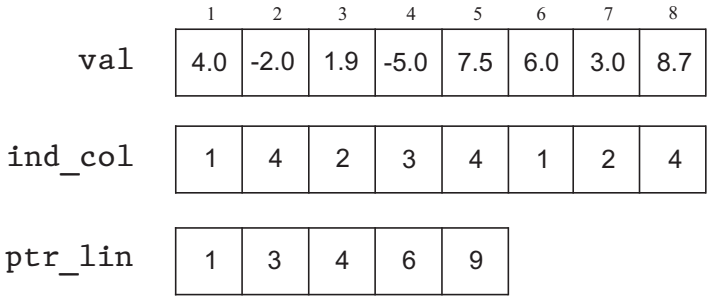

Figura 5.2: Representação em CSR da matriz M.

Como pode ser visto, os vetores `val` e `ind_col` possuem o mesmo tamanho ($m = 4$). O vetor `ptr_lin` possui dimensão $m+1$, seu último valor será sempre equivalente ao número de elementos não-nulos presentes na matriz mais uma unidade, ou seja `ptr_lin`$[m + 1] = nnz + 1$.

Os vetores `ind_col` e `ptr_lin` podem ser armazenados usando o tipo inteiro. Já o vetor `val` usando ponto flutuante. Assim, a quantidade de memória necessária para esta representação está diretamente relacionada ao nnz da matriz e sua dimensão, então usando uma representação de um *double* de 8 *bytes* e um *integer* de 4 *bytes*, seriam necessários $8nnz + 4nnz + 4(m + 1)$ *bytes* de memória.

A linhas da matriz podem ser acessadas percorrendo o vetor `ptr_lin`, assim os elementos de uma determinada linha i estarão sempre no intervalo `ptr_lin[i]` e `ptr_lin`$[i+1]-1$,

sendo $i = 1, \ldots, m$.

Operações envolvendo a representação CSR são mais complexas quando comparadas com a abordagem `densa` devido a maneira de acessar os elementos presentes na estrutura. Ter acesso aos elementos por coluna pode ser um complicador. Para este caso recomenda-se o uso da forma CSC.

5.2.1.2 Estrutura Compacta por Colunas (CSC)

Assim como na estrutura CSR, a estrutura CSC faz o uso de um conjunto composto de três vetores `val`, `ind_lin` e `ptr_col`. A análise realizada para a estrutura CSR é estendida para a estrutura CSC com a diferença de que agora os elementos nnz da matriz estão dispostos na ordem em que aparecem por colunas. Desta forma o vetor `ptr_col` agora irá armazenar as referências onde iniciam-se cada coluna na respectiva matriz.

Para ilustração, a Figura 5.3 mostra a matriz (5.3) armazenada em CSC.

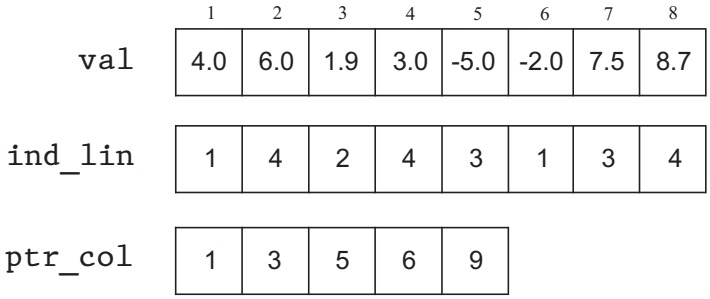

Figura 5.3: Representação em CSC da matriz M.

A quantidade de memória necessária para o armazenamento da estrutura CSC pode ser calculada usando a mesma abordagem realizada com CSR. Por conseguinte, as operações utilizando CSC também são mais complexas comparadas a forma `densa`, e podem ser realizadas de maneira eficiente quando o objetivo é acessar os elementos por colunas, bastando, de maneira análoga a CSR, apenas percorrer o vetor `ptr_col`. O acesso aos elementos por linha nesta estrutura pode ser muito custoso, portanto deve-se recorrer a estrutura CSR neste caso.

5.2.1.3 Representação da Matriz Inversa da Base com Conjunto de Vetores Compactos CSC/CSR na FPI

No presente trabalho, ao utilizar o conjunto de vetores compactos CSC/CSR na abordagem FPI, foi necessário armazenar a inversa da base utilizando a estrutura CSC fazendo com que os valores referentes a posição k fossem armazenados na primeira posição de cada coluna correspondente ao vetor η_i no vetor `val_eta_csc`, como mostrado na Figura 5.4.

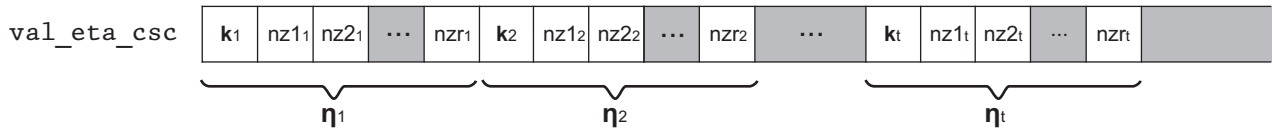

Figura 5.4: Representação do vetor `val_eta_csc`.

Como pode ser visto, após as posições k_t de cada vetor eta existem posições equivalentes ao número de elementos não-nulos (nz) de cada vetor, sendo este número representado por r elementos em cada eta. Vale notar, que ao final do vetor `val_eta_csc` existe uma área sombreada, sendo esta correspondente a posições extras reservadas para conterem o crescimento dos vetores eta durante as iterações. Diante disso, foi necessário o desenvolvimento de heurísticas de alocação de forma a sempre possuir algumas posições extras sem no entanto fazer uso desordenado da memória.

Nas implementações não foi realizado nenhum processo de *refatoração* ou *reinversão* da matriz básica com o aumento dos vetores eta. Como abordado em Munari 2009 [25], este processo consiste no descarte da representação atual e posterior obtenção de uma nova representação da inversa da matriz básica por meio de um procedimento específico.

Com a abordagem adotada na Figura 5.4, nas posições correspondentes a k no vetor `ind_lin_eta` utilizou-se o valor -1 como forma de identificador. Para demonstrar a estrutura completa, suponha 3 vetores eta com seus respectivos valores k:

$$\underset{\boldsymbol{\eta}_1\ k=1}{\begin{bmatrix} 2 \\ 3 \\ 0 \\ 4 \\ 0 \end{bmatrix}} \quad \underset{\boldsymbol{\eta}_2\ k=3}{\begin{bmatrix} 0 \\ 1 \\ 5 \\ 6 \\ 0 \end{bmatrix}} \quad \underset{\boldsymbol{\eta}_3\ k=2}{\begin{bmatrix} 0 \\ 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}}$$

A estrutura em CSC correspondente ao armazenamento dos vetores eta anteriormente, seria:

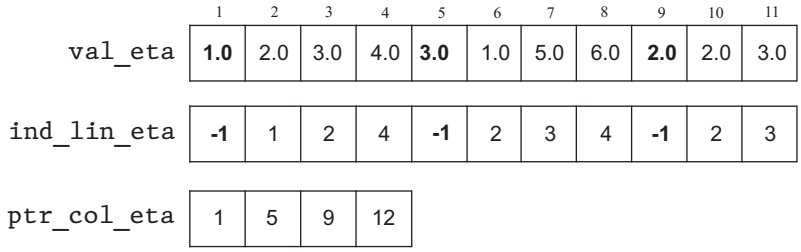

Figura 5.5: Representação de $\boldsymbol{\eta}_1$, $\boldsymbol{\eta}_2$ e $\boldsymbol{\eta}_3$ em CSC.

Com esta representação, a manipulação da estrutura CSC torna-se um pouco mais complexa, já que deve-se sempre estar atento às posições referentes a k, contudo as rotinas do DSC que envolvem o conjunto de vetores eta são beneficiadas devido sua representação esparsa. Também há o benefício de economia de memória, visto que não é necessário um vetor extra para armazenar os valores de k para cada eta.

5.2.1.4 Algoritmo de Atualização com Conjunto de Vetores Compactos CSC/CSR

Ao realizar o processo de atualização da base na Forma Revisada utilizando o conjunto de vetores compactos CSC/CSR, foram necessárias adaptações para que este pudesse ser feito de maneira otimizada. Como consequência, um algoritmo foi construído especificamente para esta etapa. Vale ressaltar que não foi encontrado na literatura tal procedimento para esta forma de atualização com as respectivas estruturas de dados.

Antes da exposição da estrutura do algoritmo criado, considere as seguintes matrizes:

$$(B^\top)^{-1} = \begin{bmatrix} \mathbf{2} & 0 & -1 & 2 \\ \mathbf{0} & -3 & 0 & 0 \\ \mathbf{0} & 0 & 5 & 0 \\ \mathbf{-1} & 0 & 2 & 1 \end{bmatrix} \qquad E^\top = \begin{bmatrix} \mathbf{-2} & \mathbf{0} & \mathbf{5} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

Em que a matriz $(B^\top)^{-1}$ representa, no DSC, a matriz transposta da inversa da base atual na iteração t e $E^\top$ a matriz transposta da matriz de transformação nesta iteração. A coluna em negrito de $(B^\top)^{-1}$ refere-se a $k = 1$, sendo esta a posição no conjunto de índices básicos (I_B) associada ao índice p básico. Por conseguinte, a linha em negrito de $E^\top$ está também associada ao valor de k, neste caso linha 1. Considere o armazenamento da matriz $(B^\top)^{-1}$ em um conjunto de vetores compactos CSC, como mostrado na Figura 5.6.

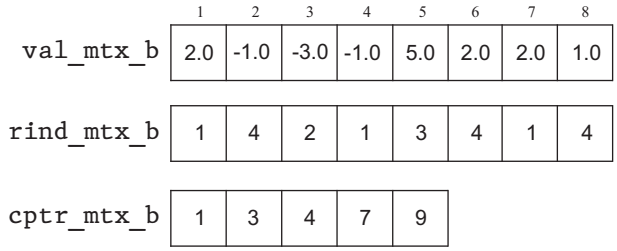

Figura 5.6: Representação de $(B^\top)^{-1}$ em CSC.

O Algoritmo 4 encontra-se em pseudocódigo de alto nível e demonstra de forma resumida o processo desenvolvido especificamente para a atualização com as estruturas CSC/CSR usando a Forma Revisada. Para este processo, a matriz de restrições técnicas do problema (N) encontra-se armazenada no formato de vetores compactos CSR, pois a maioria das operações realizadas nesta matriz são feitas por linhas, o que justifica o uso desta estrutura. A matriz $(B^\top)^{-1}(B)$ se encontra armazenada no formato CSC, sendo também escolhida esta estrutura por facilitar as operações nesta matriz. Com o uso destas estruturas, foi necessário também construir uma rotina para a obtenção do vetor correspondente a posição k da matriz $(B^\top)^{-1}$. Também se faz necessário para o cálculo da atualização, a obtenção dos valores da linha k na matriz de transformação $E^\top$.

Algoritmo 4 *Algoritmo de Atualização da Base na Forma Revisada usando CSC/CSR*

Entrada: Estruturas em CSC (val_mtx_basica, rind_mtx_basica, cptr_mtx_basica) contendo a transposta da Inversa da Base ($(B^\top)^{-1}$), posição do índice a entrar na base k, posição do índice a sair da base q, vetor com as componentes da direção dual `vdir_dual`.

Saída: Transposta da Inversa da Base ($(\overline{B}^\top)^{-1}$) atualizada.

1: *P1:* **Atualização**
   // *dim_mtx_b*: Dimensão da matriz básica
2: **para** j = 1 **até** *dim_mtx_b* **passo** 1 **faça**
3: **<BLOCO 1 DE COMANDOS>**
4: **para** i ← `cptr_mtx_basica[j]` **até** `cptr_mtx_basica[j + 1] - 1` **passo** 1 **faça**
5: **<BLOCO 2 DE COMANDOS>**
6: **fim para**
   // verifica posições restantes que são nulas
   // O valor de k1 sempre será incrementado
7: **enquanto** k1 <= *dim_mtx_b* **faça**
8: **<BLOCO 3 DE COMANDOS>**
9: **fim enquanto**
10: **fim para**
11: - Atualize o conjunto de índices básicos (I_B) e o conjunto de índices não básicos (I_N).
12: *P2:* **Atualização das soluções**
13: - Atualize a solução dual ($\overline{\lambda}$):
$\overline{\lambda}_B = \lambda_B + \delta\eta_B$
14: - Atualize a solução primal ($\overline{x}$):
$\overline{x} = \overline{B}^{-1}\overline{y}_B$

No Algoritmo 4 estão destacados 3 blocos principais de comandos, sendo que:

- **<BLOCO 1 DE COMANDOS>**: É responsável pela geração dos valores das linhas de cada coluna na estrutura CSC.
- **<BLOCO 2 DE COMANDOS>**: Percorre os valores das colunas da matriz básica atual para gerar os valores da nova matriz. Valores estes que são obtidos a partir das posições que se encontram referenciados na estrutura `cptr_mtx_basica`.
- **<BLOCO 3 DE COMANDOS>**: Gerencia os valores que podem ser gerados a partir das posições que não se encontram referenciados na estrutura `cptr_mtx_basica` devido a nulidade das mesmas.

O Algoritmo 4 se mostrou muito eficiente na prática como pode ser visto no Capítulo 6. Este desempenho se justifica pelo fato das matrizes envolvidas no cálculo possuírem alta esparsidade e das estruturas utilizadas se beneficiarem diretamente desta característica na realização das operações.

5.2.2 Representação dos dados usando listas encadeadas

Ao representar os dados com a utilização de listas encadeadas, decidiu-se que a melhor abordagem considerando as rotinas do Algoritmo DSC seria um vetor de ponteiros para listas. Esta representação foi utilizada tanto para a matriz A de restrições técnicas do problema quanto para a matriz básica B em ambas as formas de atualização da base.

O vetor de ponteiros criado representa as linhas de uma matriz, sendo que cada posição possui duas informações: o número de elementos atuais na respectiva linha (`num_elem`) e um ponteiro para o primeiro elemento (`ini`). Já na estrutura que representa os elementos em si (`nó`) existem três campos: A informação correspondente ao índice da coluna (`col`), o valor (`val`) e um ponteiro para o próximo elemento da lista (`prox`). A Figura 5.7 sumariza as informações apresentadas.

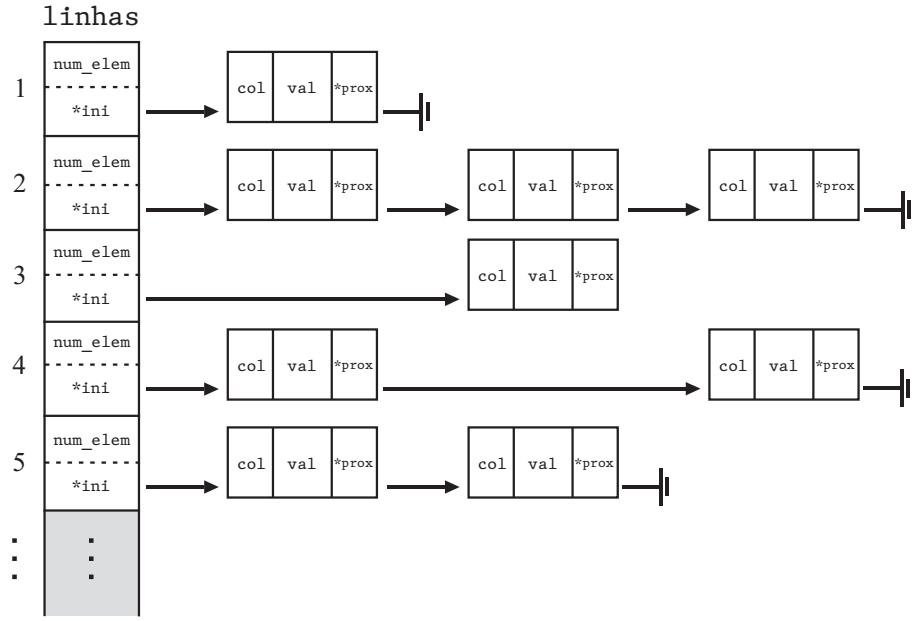

Figura 5.7: Representação utilizada nas implementações com listas encadeadas.

Esta forma de representação possui a vantagem de preservar a esparsidade dos dados, em contrapartida implica em uma ligeira complexidade na implementação para gerenciar a atualização das informações de maneira correta. Sendo assim necessário, na maioria das vezes, percorrer a estrutura para inserir, remover ou atualizar determinado elemento, o que influencia diretamente no seu desempenho.

Para representação da base na FPI foi necessária uma ligeira modificação da estrutura acima, pois como a base nesta forma de atualização é armazenada por meio de um conjunto de vetores `eta`, então isto fez com que fosse necessário mapear não mais as linhas de uma matriz, mas sim o número de `etas`. Neste caso o vetor de ponteiros armazenou o valor de k do respectivo `eta`, o número de elementos (`num_elem`) e um ponteiro (`ini`) para o primeiro elemento. A estrutura principal (`nó`) na nova abordagem passou a armazenar a informação referente a linha (`lin`), e não mais da coluna, para se adequar a forma de realizar as operações com os vetores `eta`. A Figura 5.8 mostra como ficou a estrutura após as modificações.

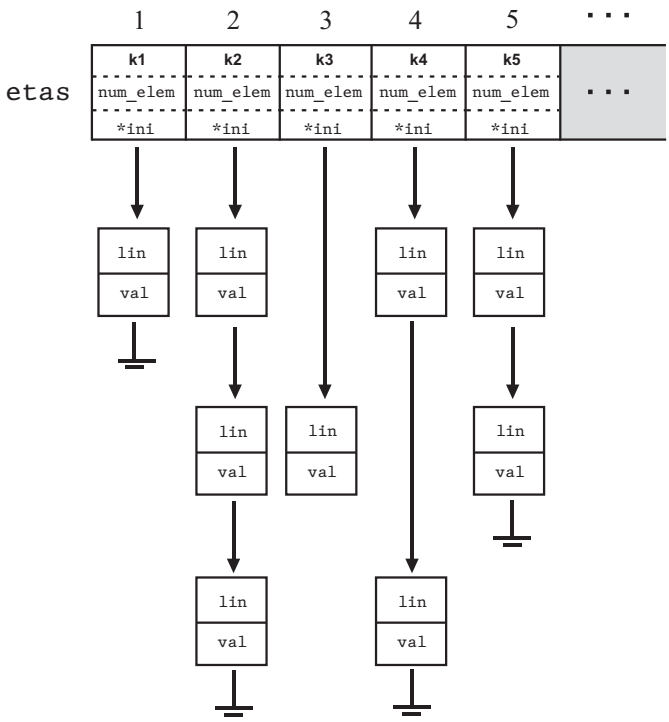

Figura 5.8: Representação utilizada para armazenamento dos `etas` com listas encadeadas.

Com esta adaptação também se fez necessária a implementação de heurísticas de alocação de forma a gerenciar o número de posições no vetor de ponteiros (vetores `eta`) alocados sem fazer o uso desordenado da memória disponível.

5.3 Implementação Computacional

Esta seção tem por objetivo abordar como as diferentes implementações do DSC foram abstraídas do ponto de vista de projeto do software, detalhando também a organização do processo de entrada de dados. Esta fase é de extrema importância, visto que nela foram decididos qual paradigma de programação que seria utilizado, divisão de responsabilidades e a escolha do formato padrão dos arquivos de entrada utilizados na validação das implementações.

5.3.1 Estrutura do Código Implementado

Durante o projeto de desenvolvimento do algoritmo buscou-se projetá-lo de modo com que alguns princípios importantes da engenharia de software fossem seguidos, tais como, modularidade, independência funcional e responsabilidade única. Estes princípios tem por objetivo tornar o processo de desenvolvimento mais ágil, organizado e gerenciável.

Ao se estruturar o software levando em consideração as características citadas, tem-se como consequência o impacto direto no produto final desenvolvido, visto que a manutenção do código é facilitada, tornando possível modificações futuras com baixo impacto nos demais módulos componentes, fazendo com que se tenha um baixo acoplamento. Outro benefício é que, ao se ter uma independência funcional entre os módulos, partes do software podem ser utilizados em outros projetos onde, a princípio, não possuem uma ligação direta.

Quando se trata do desenvolvimento de sistemas para solução de problemas de PL, Maros [34] destaca algumas propriedades importantes e que devem ser consideradas. Segundo o autor, algumas destas propriedades podem ser asseguradas durante o projeto do algoritmo e outras durante a fase de implementação de acordo com a Tabela 5.1.

Tabela 5.1: Propriedades desejáveis em um software de PL.

	Projeto do Algoritmo	**Implementação**
Robustez	X	X
Acurácia	X	X
Eficiência	X	X
Capacidade	X	X
Portabilidade		X
Modularidade		X
Estabilidade do código		X
Manutenabilidade		X
Escalabilidade		X
Usabilidade		X

Com o objetivo de se atingir o máximo das propriedades recomendas na Tabela 5.1, neste trabalho buscou-se desenvolver o software utilizando o paradigma de programação orientado a objetos (OO) na linguagem de programação C++. Com a utilização do paradigma OO foi possível abstrair e organizar o código em classes de acordo com o diagrama mostrado na Figura 5.9.

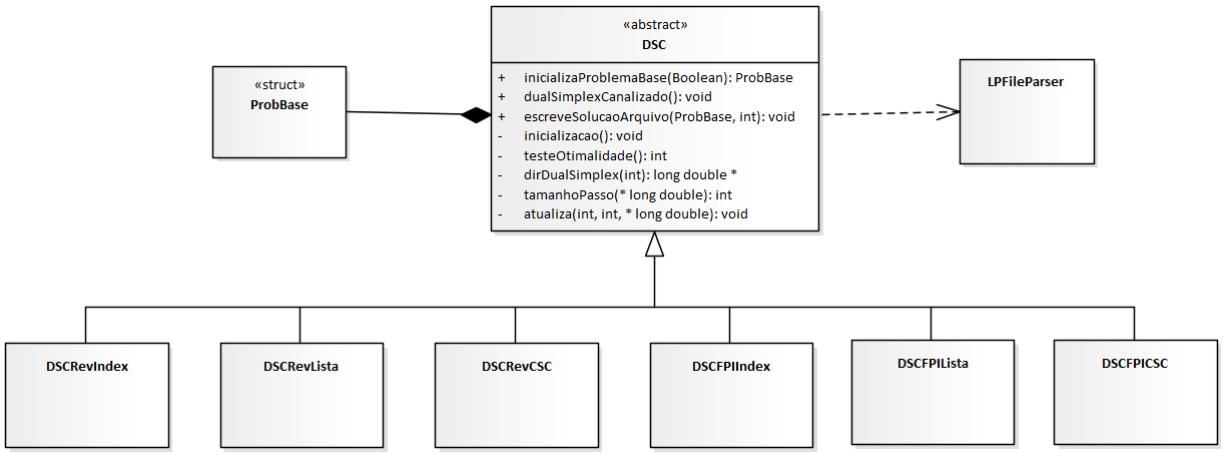

Figura 5.9: Diagrama de Classes das implementações realizadas.

Como pode ser visto na Figura 5.9, a classe `DSC` tem por finalidade ser a classe base (classe mãe) de todas as implementações realizadas (classes filhas), visto que todas as implementações compartilham das mesmas características, ou seja, são implementações do algoritmo DSC apresentado no Capítulo 3. O que as diferencia é que cada classe possui uma implementação com uma estrutura de dados específica.

Com a organização acima exposta, foi possível centralizar as caraterísticas em comum de cada implementação, e ao mesmo tempo diferenciá-las tomando por base as diferentes estruturas de dados propostas para a validação do código. Também foi possível gerenciar cada implementação de maneira individual, de modo que modificações em uma determinada implementação não impactasse de maneira direta e descontrolada em outra.

Uma característica importante do paradigma OO que pôde ser utilizada pela adoção da estrutura apresentada é o polimorfismo. Esta característica é fundamental para tratar de maneira homogênea as implementações sem levar em consideração em "como" cada classe implementa suas características particulares, mas sim "o que" cada uma delas implementa.

O polimorfismo é alcançado através da definição de um tipo básico, que na na Figura 5.9 é caracterizado pela classe principal `DSC`, e por meio de interfaces bem definidas, descritas por seus métodos. Cada classe filha sobrescreve os métodos nos quais irão se diferenciar da classe mãe.

Como também pode ser observado, todas as implementações fazem o uso de um tipo básico definido como `ProbBase`. Este tipo tem por objetivo armazenar todos os atributos

comuns de um problema lido do arquivo de entrada no formato adotado. A Tabela 5.2 descreve estes atributos compartilhados com todas as implementações realizadas.

Tabela 5.2: Atributos de um problema compartilhados com todas as implementações.

Nome Atributo	**Significado**
valor_fo	Valor da Função Objetivo
m	Número de restrições
n	Número de varáveis
nnz	Número de elementos não nulos das restrições
vc	Vetor de custos
lb	Vetor com os valores de *lower bound* das restrições
ub	Vetor com os valores de *upper bound* das restrições
ib	Vetor com os índices pertencentes a base
inb	Vetor com os índices não pertencentes a base

O arquivo lido é carregado em memória por um componente desenvolvido especificamente para leitura dos dados de entrada, representado no diagrama como `LPFileParser`. Detalhes do formato escolhido para entrada de dados bem como o *parser* dos arquivos são abordados na próxima seção.

O código implementado realiza a leitura das instâncias por um diretório pré-definido e também grava os arquivos de solução em diretórios de maneira separada de acordo com o tipo de estrutura de dados utilizada na implementação. A Figura 5.10 mostra parte de um arquivo de solução gerado pela implementação do DSC com a FPI e fazendo o uso de vetores esparsos CSC e CSR.

```
-------- Método DUAL SIMPLEX CANALIZADO (DSC) - IMPLEMENTAÇÃO FPI CSC  --------
-------------------- DADOS DO PROBLEMA:
Nome da instância.....................: instancias/SHIP04L.lp
Número de restrições..................: 353
Número de variáveis...................: 2118
Não-Nulos.............................: 6311
Esparsidade...........................: 99.16%
----------------- PARÂMETROS DO SOLVER:
Representação padrão para +inf........: 999999
Tol. violação lim. sup. inf...........: 1.00e-08
Tol. comparação com 0.................: 1.00e-09
Tol. degeneração .....................: 1.00e-08
--------------------- DADOS DA SOLUCAO:
Número de iterações...................: 396
Tempo Teste ótimo.....................: 0.00200 seg. 6.45%
Tempo Direção dual....................: 0.00300 seg. 9.68%
Tempo Tamanho do passo................: 0.00300 seg. 9.68%
Tempo Atualização.....................: 0.02200 seg. 70.97%
Tempo de execução.....................: 0.03100 seg. 100%
Valor da FO...........................: 1.7933245380e+06
Status da solução.....................: otimo
Valor das variáveis...................:
X1 = 187.378927
X2 = 28.642675
X3 = 34.631434
X4 = 11.157328
```

Figura 5.10: Arquivo de solução parcial DSCFPICSC - Problema `SHIP04L.lp` da NetLib.

5.3.2 Entrada de Dados

Para que seja possível resolver um problema de PL em meio computacional, inicialmente é necessário que este seja carregado em memória e esteja disponível em estruturas de dados que permitam sua posterior manipulação de maneira correta e eficiente. Muitos *solvers* comerciais, tais como CPLEX[1] e Gurobi[2], possuem *parsers* embutidos que têm por finalidade realizar a leitura do arquivo de entrada em um determinado formato e organizá-lo em memória para posterior solução.

5.3.2.1 Formato MPS e LP

Durante o projeto de desenvolvimento do código foi necessário definir um formato padrão dos arquivos de entrada de dados. Arquivos de instâncias de problemas reais disponíveis em repositórios da *internet*, geralmente se encontram em um formato conhecido por MPS.

O formato de arquivos MPS remonta da época dos antigos mainframes e cartões perfurados e está orientado por colunas. Este formato, como destacado por Kuip [8], caracteriza-se por ser de difícil compreensão e manutenção, por consequência seu mapeamento para a forma algébrica, e vice versa, torna-se trabalhoso e pouco legível. Detalhes deste formato estão descritos no Apêndice A.

[1] `https://www.ibm.com/analytics/cplex-optimizer`
[2] `http://www.gurobi.com/index`

Com o objetivo de uniformizar a entrada de dados para o código implementado visando maior legibilidade na representação das instâncias a serem testadas, visto que foi necessário realizar testes para validar junto as implementações, ficou definido que o formato de arquivo conhecido como LP iria ser utilizado.

No formato de arquivo LP os problemas são descritos na forma algébrica, onde é possível defini-lo de uma maneira bem próxima da qual estamos acostumados a fazer na Matemática usual, sendo este um facilitador em relação a aspectos de *debugging* por exemplo. Os detalhes do formato de arquivo LP podem ser encontrados no Apêndice B.

Com a escolha do formato LP, foi necessário então adequar a este padrão todas as instâncias dos problemas utilizados para validação do código. Como anteriormente citado, a grande maioria das instâncias de problemas de PL presentes nos repositórios da *internet* estão no formato MPS. Em particular as instâncias do repositório NetLib que foram utilizadas como parte da validação do código. Para padronização das instâncias ao formato LP, um conversor, chamado de `MPSToLPConverter` foi desenvolvido utilizando a linguagem de programação C++.

O conversor funciona como um módulo à parte, bastando apenas informar o nome do arquivo no formato MPS e então, o arquivo de saída no formato LP, com o mesmo nome do arquivo de entrada, será gerado e salvo em um diretório específico pré-definido. Maros [34] destaca que o processamento de um arquivo no formato MPS é, surpreendentemente, um procedimento complicado.

No processo de geração do arquivo LP, o `MPSToLPConverter` realiza a renomeação das variáveis, a qual é feita atribuindo índices a cada variável de maneira crescente, sendo $x_1 \dots x_n$, onde n corresponde ao número de variáveis do problema lido. Este processo é de extrema importância, pois tem por objetivo dar mais clareza ao problema lido e facilitar sua leitura posterior.

É importante destacar que o `MPSToLPConverter` atualmente não realiza conversão de arquivos MPS que possuem marcadores de variáveis inteiras (`MARKER e INTORG`), visto que a implementação do algoritmo `DSC` não lida com problemas inteiros. A Figura 5.11 mostra um exemplo de arquivo no formato MPS e, em seguida, na Figura 5.12 é apresentado o arquivo LP correspondente gerado pelo `MPSToLPConverter`.

```
NAME              ExemploMPS
ROWS
 N  COST
 L  LIM1
 G  LIM2
 E  EQN
COLUMNS
    VR...01       COST                 1.0   LIM1                 1.0
    VR...01       LIM2                 1.0
    VR...02       COST                 4.0   LIM1                 1.0
    VR...02       EQN                 -1.0
    VR...03       COST                 9.0   LIM2                 1.0
    VR...03       EQN                  1.0
RHS
    RHS1          LIM1                 5.0   LIM2                10.0
    RHS1          EQN                  7.0
BOUNDS
 UP BND1          VR...01              4.0
 LO BND1          VR...02             -1.0
 UP BND1          VR...02              1.0
 FR BND1          VR...03
ENDATA
```

Figura 5.11: Arquivo no formato MPS.

```
MIN
OBJ: 1 X1 +4 X2 +9 X3
ST
1 X1 +1 X2 <= 5
1 X1 +1 X3 >= 10
-1 X2 +1 X3 = 7
BOUNDS
X1 <= 4
X2 >= -1
X2 <= 1
X3 free
END
```

Figura 5.12: Arquivo no formato LP gerado pelo `MPSToLPConverter`.

O *solver* comercial CPLEX 12.8 também possui um conversor de arquivos embutido, onde é possível ler um arquivo de entrada no formato MPS e obter o arquivo de saída equivalente em vários formatos, como por exemplo, o formato LP. A Figura 5.13 mostra o mesmo arquivo da Figura 5.11 convertido para o formato LP com este recurso do CPLEX 12.8. Como pode ser visto, a ferramenta não realiza a nomeação das variáveis no processo de conversão, o que acaba prejudicando sua legibilidade e por este motivo, este recurso não foi utilizado no código implementado.

```
1  Minimize
2   COST: VR...01 + 4 VR...02 + 9 VR...03
3  Subject To
4   LIM1#0: VR...01 + VR...02 <= 5
5   LIM2#1: VR...01 + VR...03 >= 10
6   _EQN#2: - VR...02 + VR...03  = 7
7  Bounds
8   0 <= VR...01 <= 4
9  -1 <= VR...02 <= 1
10       VR...03 Free
11 End
```

Figura 5.13: Arquivo no formato LP gerado pelo CPLEX 12.8.

Após o desenvolvimento do conversor, restava então codificar um *parser*, tendo este a função de analisar os arquivos LP gerados e carregá-los na memória de maneira uniforme para todas as implementações realizadas. Este *parser* foi desenvolvido também como um componente a parte, denominado `LPFileParser`.

A classe `DSC` apresentada na seção anterior faz o uso do componente `LPFileParser` de maneira transparente das demais implementações. A grande vantagem de se ter este componente modularizado reside no fato de que o mesmo pode ser utilizado em outros projetos que precisam lidar com o formato de arquivo LP e possuir suas informações organizadas de maneira estruturada em memória.

Ao realizar a conversão dos arquivos MPS para o formato LP, a ordem das variáveis podem ser modificadas, visto que estas são renomeadas neste processo. Na prática este detalhe não muda o problema original, mas o caminho percorrido pelo algoritmo até encontrar a solução poderá diferir do caminho que ele percorreria originalmente.

Vale ressaltar que o arquivo mostrado na Figura 5.11 no formato MPS, ocupa em disco 702 *bytes*, enquanto o arquivo da Figura 5.12 ocupa exatos 135 *bytes*, uma redução de tamanho em disco de mais de 80%. Reduções no tamanho em disco foram observadas em todas as instâncias convertidas do repositório da NetLib. Os gráficos das Figuras 5.14 e 5.15 mostram as reduções em disco de algumas instâncias convertidas.

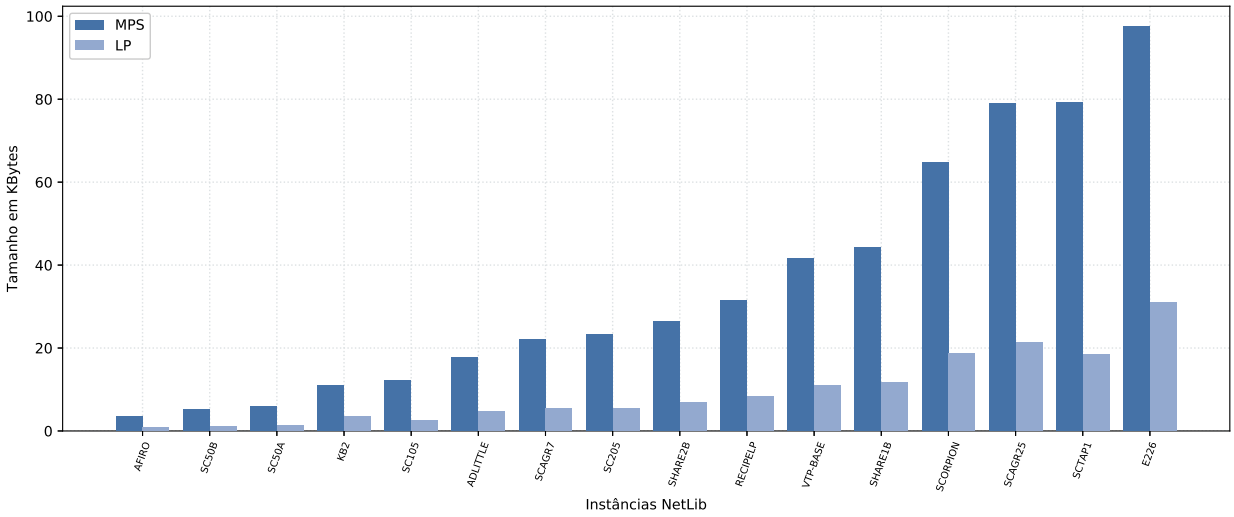

Figura 5.14: Comparativo tamanho das instâncias da NetLib - Parte 1.

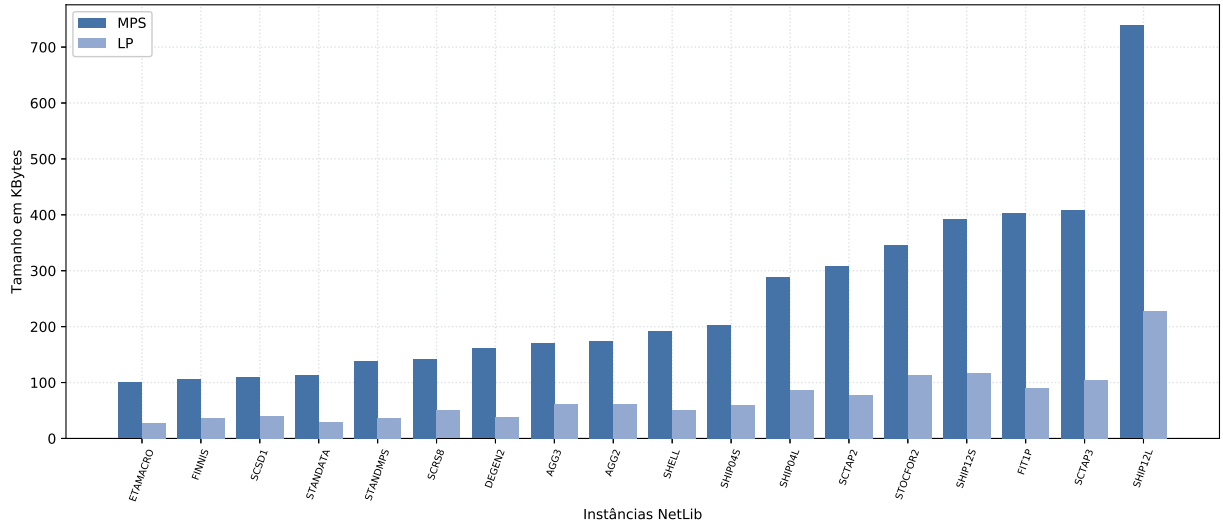

Figura 5.15: Comparativo tamanho das instâncias da NetLib - Parte 2.

O `LPFileParser` em sua implementação atual é capaz de ler arquivos LP na forma canalizada, tanto nas restrições quanto nas variáveis. O *parser* possui tratamento de exceções para eventuais erros que possam ocorrer durante a leitura de um arquivo. Também é possível ver o passo a passo da leitura de um arquivo, bastando o usuário indicar por meio de um parâmetro esta opção.

Com relação as instâncias do gerador de problemas esparsos de produção, este também teve que ser adaptado para a construção dos arquivos no formato LP. As instâncias relacionadas ao problema da mistura da fábrica de ração do IFES foram elaboradas com base nos relatórios coletados em visitas realizadas ao Campus da Instituição e foram adap-

tadas manualmente para ficarem no formato LP. Esta adaptação foi possível porque as instâncias possuíam tamanhos relativamente pequenos, mas importantes para validação do código sobre o ponto de vista numérico.

Para o desenvolvimento do conversor `MPSToLPConverter` e do *parser* `LPFileParser` foi necessário realizar um estudo das particularidades de cada formato de arquivo (MPS e LP), o que naturalmente demandou um certo tempo para que todo o processo do tratamento de entrada de dados ficasse de maneira satisfatória.

O diagrama da Figura 5.16 tem por objetivo ilustrar como o processo de entrada de dados foi estruturado.

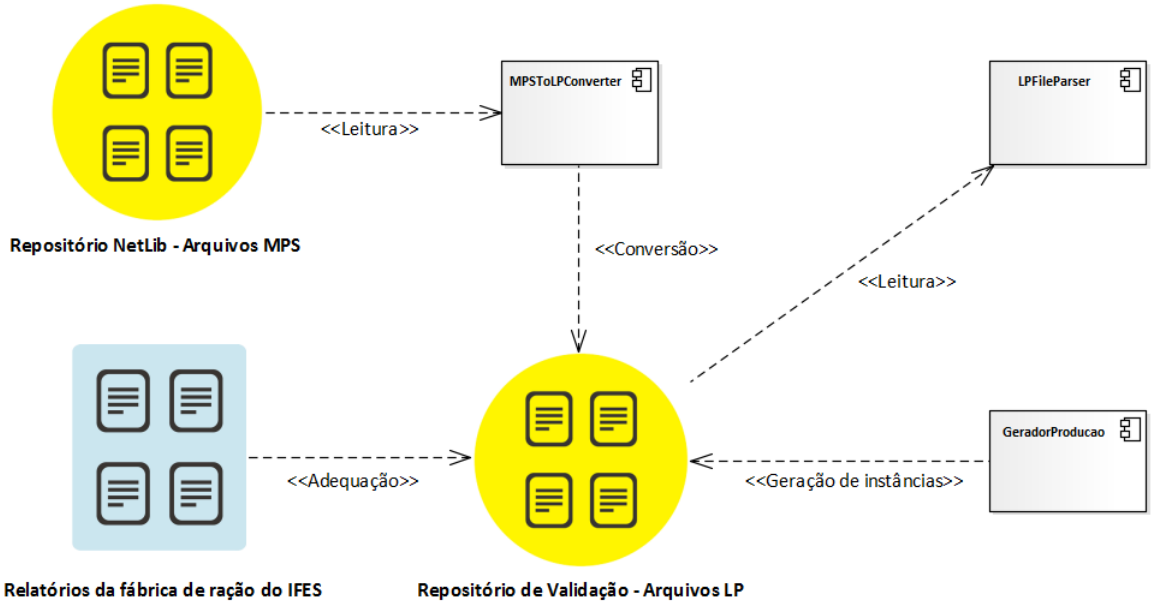

Figura 5.16: Estrutura do processo de entrada de dados.

Como pode ser visto na Figura 5.16, todas as instâncias são uniformizadas para o formato LP e armazenadas em um diretório padrão (Repositório de Validação). Posteriormente, o componente `LPFileParser` realiza a leitura dos dados contidos no arquivo de instância especificado e os carrega em estruturas de dados específicas onde posteriormente poderão ser utilizados por uma das implementações realizadas.

Capítulo 6

Resultados Computacionais

Neste capítulo são apresentadas as classes de problemas tratados com suas respectivas instâncias e a análise dos resultados computacionais do método DSC implementado com as três estruturas de dados e as duas formas de atualização da base.

6.1 Instâncias Utilizadas nos experimentos computacionais

Para a realização dos testes computacionais foram utilizadas 3 classes de problemas distintas contendo: 15 instâncias de uma fábrica de ração, 36 instâncias com alto índice de esparsidade geradas de forma pseudoaleatória relacionadas a problemas de programação da produção multiperíodo e 31 instâncias do repositório da NetLib.

Tomando por base uma instância de um problema onde m representa o número de restrições técnicas e n o número de variáveis, estas foram classificadas pelo autor em relação a dimensão do número de variáveis, como sumarizado na Tabela 6.1.

Tabela 6.1: Classificação das instâncias

Classificação	Nº de variáveis
Pequenas	$n \leq 1000$
Médias	$1000 < n < 5000$
Grandes	$n \geq 5000$

Nas seções seguintes, cada classe de problemas das quais as instâncias utilizadas pertencem, são contextualizadas. Para cada uma destas classes, as informações apresentadas de suas instâncias são: nome da instância, dimensões (linhas e colunas), número de não-nulos e esparsidade. Sendo esta última representada de forma percentual em termos do número de elementos nulos presentes.

6.1.1 Instâncias da Fábrica de Ração do IFES

A formulação de rações configura-se como um problema de mistura. O problema consiste na combinação de alimentos em proporções que atendam às demandas nutricionais de animais a um custo mínimo. Para isto, a mistura deve ser feita de maneira que o produto final atinja certos níveis para diferentes tipos de nutrientes, como proteínas, cálcio, carboidratos, vitaminas, etc; Bazaraa(2010) [3].

Segundo Rostagno [38], para se calcular rações de custo mínimo utilizando a metodologia de PL são necessárias as seguintes informações: preços dos alimentos, alimentos disponíveis, composição dos alimentos, exigências nutricionais dos animais e restrições ou limitantes dos alimentos.

A motivação para testar instâncias deste tipo de problema reside no fato de que o problema da mistura pode ser modelado de forma canalizada nas restrições e variáveis, sendo este tipo de modelagem ideal para testes com o método DSC implementado.

Para a obtenção dos dados na formulação das instâncias do problema da mistura, foram realizadas algumas visitas a fábrica de ração do Instituto Federal de Educação Ciência e Tecnologia do Espírito Santo - Campus de Alegre (IFES - Campus Alegre[1]). Nestas visitas, por meio da autorização do zootecnista responsável, as informações necessárias foram coletadas através de relatórios de formulação de rações para aves já existentes e utilizados no Instituto.

No processo de formulação, para a obtenção das informações de composição dos alimentos e das exigências nutricionais, foram utilizados os dados presentes nas Tabelas Brasileiras para Aves e Suínos [37] publicada pelo Departamento de Zootecnia da Universidade Federal de Viçosa - UFV e utilizada como referência no país para formulação de rações.

A Tabela 6.2 mostra as informações das 15 instâncias elaboradas. Todas as instâncias são classificadas como Pequenas, mas são de grande importância, pois representam mo-

[1] `https://alegre.ifes.edu.br/`

delos de problemas reais utilizados na fábrica.

Tabela 6.2: Instâncias do problema da mistura da Fábrica de ração.

Instância	**Linhas**	**Colunas**	**Não-Nulos**	**Esparsidade** (%)
inst1_prob_mistura	22	9	61	69,19
inst2_prob_mistura	22	9	60	69,70
inst3_prob_mistura	24	9	67	68,98
inst4_prob_mistura	25	9	73	67,56
inst5_prob_mistura	22	9	62	68,69
inst6_prob_mistura	25	9	73	67,56
inst7_prob_mistura	22	9	62	68,69
inst8_prob_mistura	25	9	73	67,56
inst9_prob_mistura	22	6	53	59,85
inst10_prob_mistura	23	10	75	67,39
inst11_prob_mistura	23	11	86	66,01
inst12_prob_mistura	28	11	116	62,34
inst13_prob_mistura	27	11	113	61,95
inst14_prob_mistura	26	10	84	67,69
inst15_prob_mistura	39	11	152	64,57

6.1.2 Instâncias do Gerador de Problemas de Programação da Produção

Outra classe de problema utilizada nos testes consiste em um modelo que simula a programação da produção multiperíodo. Este tipo de modelo pode ser formulado levando em consideração a produção de um ou mais produtos com previsão de demandas em determinados períodos, sendo que deve-se determinar quanto produzir e estocar durante o período a fim de minimizar os custos de produção e estocagem, Rangel(2012) [36]. O problema pode ser formulado conforme mostrado em (6.1).

$$
\begin{aligned}
Minimizar \quad & \sum_{i=1}^{p}\sum_{t=1}^{T} c_{it}x_{it} + \sum_{i=1}^{p}\sum_{t=1}^{T} h_{it}e_{it} & & \\
Sujeito\ a \quad & \sum_{i=1}^{p} b_i x_{it} \leq k_t & & t = 1, \ldots T \\
& e_{it-1} + x_{it} - e_{it} = d_{it} & & i = 1, \ldots, p \ \ e \ \ t = 1, \ldots, T \\
& x_{i,t} \geq 0, e_{it} \geq 0, d_{it} \geq 0 & &
\end{aligned}
\tag{6.1}
$$

Onde:

- p: Quantidade de produtos em T períodos com demandas d_{it};

- k_t: Capacidade de uma máquina no período t;
- h_{it}: Custo para estocar uma unidade de i no período t;
- b_i: Quantidade de recurso usado da máquina na fabricação de uma unidade de i;
- c_{it}: Custo para produzir uma unidade de i no período t;
- x_{it}: Quantidade a ser produzida de i no período t;
- e_{it}: Quantidade a ser estocada de i no período t.
- d_{it}: Demanda do produto i no período t.

Ressalta-se que na prática, problemas descritos como no modelo apresentado, devem ser tratados como problemas inteiros e com outras possíveis restrições, Rangel(2012) [36]. Portanto, os resultados apresentados pelo DSC não garantem a integralidade das variáveis.

Instâncias oriundas de modelos de produção apresentam alta esparsidade e os elementos da matriz A se encontram geralmente organizados em um padrão conhecido como forma escada, Silva(2002) [9]. A Figura 6.1 mostra o padrão de esparsidade da instância inst6_prod_310x670 gerada pelo gerador pseudoaleatório com 60 itens e 5 períodos e que também foi utilizada nos testes computacionais.

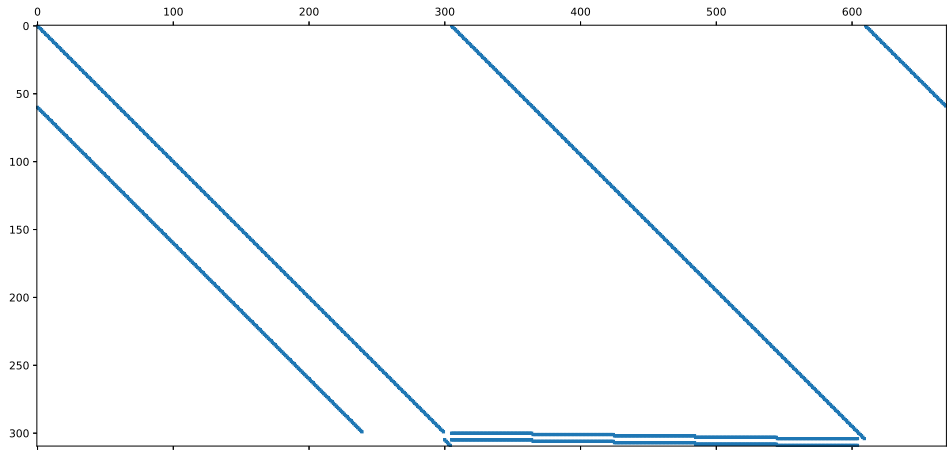

Figura 6.1: Padrão de esparsidade inst6_prod_310x670 - Dimensão = 310x670 Não-Nulos = 1510.

Os dados das instâncias geradas do modelo de programação da produção utilizadas nos testes são apresentados na Tabela 6.3.

Tabela 6.3: Instâncias do gerador de problemas de programação da produção.

Instância	**Linhas**	**Colunas**	**Não-Nulos**	**Esparsidade** (%)
inst1_prod_60x120	60	120	260	96,39
inst2_prod_110x230	110	230	510	97,98
inst3_prod_160x340	160	340	760	98,60
inst4_prod_210x450	210	450	1010	98,93
inst5_prod_260x560	260	560	1260	99,13
inst6_prod_310x670	310	670	1510	99,27
inst7_prod_360x780	360	780	1760	99,37
inst8_prod_410x890	410	890	2010	99,45
inst9_prod_460x1000	460	1000	2260	99,51
inst10_prod_510x1110	510	1110	2510	99,56
inst11_prod_800x1580	800	1580	3800	99,70
inst12_prod_900x1660	900	1660	3900	99,74
inst13_prod_1050x2090	1050	2090	5050	99,77
inst14_prod_1200x2210	1200	2210	5200	99,80
inst15_prod_1300x2600	1300	2600	6300	99,81
inst16_prod_1550x3110	1550	3110	7550	99,84
inst17_prod_1800x3620	1800	3620	8800	99,86
inst18_prod_2050x4130	2050	4130	10050	99,88
inst19_prod_2200x4220	2200	4220	10200	99,89
inst20_prod_2300x4640	2300	4640	11300	99,89
inst21_prod_2550x5150	2550	5150	12550	99,90
inst22_prod_3150x6190	3150	6190	15150	99,92
inst23_prod_3600x7170	3600	7170	17600	99,93
inst24_prod_3720x7380	3720	7380	18120	99,93
inst25_prod_3900x7700	3900	7700	18900	99,94
inst26_prod_4100x8180	4100	8180	20100	99,94
inst27_prod_4600x9190	4600	9190	22600	99,95
inst28_prod_5100x10200	5100	10200	25100	99,95
inst29_prod_6120x12220	6120	12220	30120	99,96
inst30_prod_6900x13740	6900	13740	33900	99,96
inst31_prod_7200x14270	7200	14270	35200	99,97
inst32_prod_8200x16280	8200	16280	40200	99,97
inst33_prod_9200x18290	9200	18290	45200	99,97
inst34_prod_10200x20300	10200	20300	50200	99,98
inst35_prod_11235x22365	11235	22365	55335	99,98
inst36_prod_13260x26360	13260	26360	65260	99,98

6.1.3 Instâncias do Repositório NetLib

O repositório da NetLib contém instâncias de problemas relacionados a computação numérica, científica e afins. Atualmente a iniciativa é mantida pela AT&T Bell Laboratories, a Universidade do Tennessee e o Oak Ridge National Laboratory, entre outras

parcerias ao redor do mundo, segundo citado na seção FAQ do site[2].

As instâncias são disponibilizadas de maneira gratuita e representam modelos obtidos de problemas reais de diversas áreas, tais como: planejamento florestal, refinaria de petróleo, escalonamento de tripulações, produção industrial, etc. Desta forma, são amplamente usados na validação de softwares e pesquisas científicas que envolvem otimização linear e inteira.

Já na década de 90 ressaltava-se que muitos dos problemas contidos no repósitório, estavam lá justamente por causarem dificuldades para serem solucionados por meio de métodos do tipo simplex, Bixby [7]. Tais dificuldades estão associadas a instabilidade numérica, não escalabilidade, degeneração, etc. Ainda hoje, o repositório é utilizado para validações e testes de *Benchmarks* entre diferentes softwares de otimização[3].

Como as instâncias representam modelos de problemas reais, então a esparsidade também é uma das características determinantes. A Figura 6.2 mostra o padrão de esparsidade da instância SHIP12S[4] do repositório NetLib.

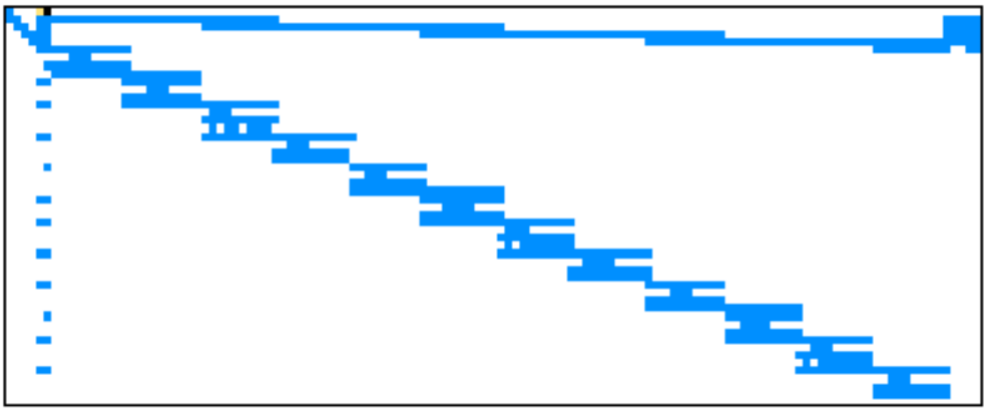

Figura 6.2: Padrão de esparsidade problema SHIP12S da NetLib - Dimensão = 1151x2763 Não-Nulos = 8178.

Na Tabela 6.4 são apresentados os detalhes das 31 instâncias do repositório da NetLib utilizadas nos testes computacionais. Os valores ótimos apresentados na tabela estão de acordo com o trabalho de Koch [31] descritos com uma precisão de 11 dígitos. Para o cálculo destes valores, com o intuito de provar a factibilidade e otimalidade das soluções dos problemas, Koch utilizou uma abordagem de aritmética racional e a biblioteca de multiprecisão aritmética GNU GMP[5].

[2] http://www.netlib.org/misc/faq.html
[3] http://plato.asu.edu/sub/testcases.html, http://plato.asu.edu/bench.html
[4] https://www.cise.ufl.edu/research/sparse/matrices/LPnetlib/lp_ship12s.html
[5] https://gmplib.org/

Tabela 6.4: Instâncias do repositório da NetLib.

Nº	Instância	Linhas	Colunas	Não-Nulos	Esparsidade (%)	Valor ótimo
1	AFIRO	27	32	83	90,39	-4,6475314286E+02
2	KB2	43	41	286	83,78	-1,7499001299E+03
3	SC50A	50	48	130	94,58	-6,4575077059E+01
4	SC50B	50	48	118	95,08	-7,0000000000E+01
5	ADLITTLE	56	97	383	92,95	2,2549496316E+05
6	SCSD1	77	760	2388	95,92	8,6666666743E+00
7	RECIPELP	91	180	663	95,95	-2,6661600000E+02
8	SHARE2B	96	79	694	90,85	-4,1573224074E+02
9	SC105	105	103	280	97,41	-5,2202061212E+01
10	SHARE1B	117	225	1151	95,63	-7,6589318579E+04
11	SCAGR7	129	140	420	97,67	-2,3313898243E+06
12	VTP-BASE	198	203	908	97,74	1,2983146246E+05
13	SC205	205	203	551	98,68	-5,2202061212E+01
14	E226	223	282	2578	95,90	-1,8751929066E+01
15	SCTAP1	300	480	1692	98,83	1,4122500000E+03
16	STANDATA	359	1075	3031	99,21	1,2576995000E+03
17	SCORPION	387	358	1426	98,97	1,8781248227E+03
18	ETAMACRO	400	688	2409	99,12	-7,5571523337E+02
19	SHIP04S	402	1458	5810	99,01	1,7987147004E+06
20	SHIP04L	402	2118	6332	99,26	1,7933245380E+06
21	DEGEN2	444	534	3978	98,32	-1,4351780000E+03
22	STANDMPS	467	1075	3679	99,27	1,4060175000E+03
23	SCAGR25	471	500	1554	99,34	-1,4753433061E+07
24	SCRS8	490	1169	3182	99,44	9,0429695380E+02
25	FINNIS	497	614	2310	99,24	1,7279106560E+05
26	AGG2	516	302	4284	97,25	-2,0239252356E+07
27	AGG3	516	302	4300	97,24	1,0312115935E+07
28	SHELL	536	1775	3556	99,63	1,2088253460E+09
29	SCTAP2	1090	1880	6714	99,67	1,7248071429E+03
30	SHIP12S	1151	2763	8178	99,74	1,4892361344E+06
31	SHIP12L	1151	5427	16170	99,74	1,4701879193E+06

6.2 Experimentos Computacionais

Nas próximas seções são apresentados os resultados computacionais realizados das 3 classes de problemas anteriormente citadas.

Para a compilação do código implementado foi utilizado o Microsoft Optimizer C/C++ Compiler V19 juntamente com a IDE Microsoft Visual Studio Community 2017 com a licença gratuita estudantil. Todos os experimentos computacionais foram realizados em um único computador Notebook com processador Intel Core I7 2.0 GHz, 12 GB de RAM e sistema operacional Windows 7 de 64 bits.

Em cada uma das 3 classes de problemas foram utilizadas as formas de atualização da base tratadas: Revisada e FPI. Para cada forma de atualização, foram testadas as 3 estruturas de dados implementadas: Indexada (Index), Lista Encadeada (LE) e conjunto de vetores compactos (CSC/CSR).

Inicialmente, os resultados analisados nos testes foram: número de iterações (IT) e tempo total de solução (Tempo Sol.). Posteriormente, para algumas instâncias classificadas como grandes dos problemas de programação da produção foram também analisados: Tempo por iteração e o tempo gasto para realizar as principais rotinas do Algoritmo1 DSC, sendo estas: Teste ótimo (Teste Ot.), Cálculo da direção dual simplex (Calc. Dir. Dual), Cálculo do tamanho do passo (Calc. Tam. Passo) e Atualização da Base (Atual. Base).

Dentre as classes de problemas testados, alguns resultados foram comparados com a implementação do algoritmo Dual Simplex presente no *solver* CPLEX 12.8. Ressalta-se que alguns parâmetros padrão do *solver* CPLEX 12.8 que possibilitam realizar pré-processamento, inicialização avançada, mudança de escala, regra de escolha de variáveis, entre outros, foram modificados. Para detalhes destes parâmetros consulte o Apêndice C.

Munari(2013) [24] destaca que muito dos desenvolvimentos teóricos e computacionais que são utilizados para melhorar a performance de *solvers* comerciais não são reportados na literatura. Como consequência, muitos autores são desencorajados a testar novas estratégias, pois vêem que o desempenho de suas implementações ficam muito distantes dos *solvers* comerciais. Assim sendo, a realização das comparações do DSC em relação ao algoritmo Dual Simplex presente no CPLEX 12.8 aqui descritas não têm por objetivo configurar a implementação do DSC como um algoritmo competitivo em relação ao algoritmo contido no *solver* CPLEX, pois este é um produto comercial com décadas de desenvolvimento. Os testes comparativos têm como finalidade mostrar que os resultados obtidos na implementação do DSC, para as classes de problemas escolhidas, são satisfatórios e contribuem para destacar que algumas estruturas de dados são mais eficientes que outras quando se implementam métodos do tipo simplex.

6.2.1 Problemas da Fábrica de Ração

Na realização dos testes com as instâncias do problema da fábrica de ração, os tempos de solução apresentaram valores inferiores a 10^{-3} segundos. Desta forma, o único resultado analisado foi o número de iterações. A Tabela 6.5 mostra os resultados com atualização da base na forma Revisada em cada uma das estruturas de dados. Para efeito de comparação, juntamente aos resultados do DSC, também se encontra na última coluna o número de

iterações que a implementação Dual Simplex contida no CPLEX 12.8 realizou para resolver os mesmos problemas.

Convém destacar que o número de iterações em ambas as formas de atualização da base para o DSC foram os mesmos para os problemas testados, por este motivo, apenas os resultados da forma Revisada são apresentados.

Tabela 6.5: Número de Iterações - DSC Revisado e CPLEX

Instância	**Index**	**LE**	**CSC/CSR**	**CPLEX**
inst1_prob_mistura	9	9	9	6
inst2_prob_mistura	9	9	9	6
inst3_prob_mistura	13	13	13	6
inst4_prob_mistura	12	12	12	7
inst5_prob_mistura	8	8	8	6
inst6_prob_mistura	12	12	12	6
inst7_prob_mistura	11	11	11	7
inst8_prob_mistura	13	13	13	7
inst9_prob_mistura	8	8	8	5
inst10_prob_mistura	1	1	1	1
inst11_prob_mistura	1	1	1	1
inst12_prob_mistura	10	10	10	9
inst13_prob_mistura	5	5	5	**8**
inst14_prob_mistura	11	11	11	7
inst15_prob_mistura	10	10	10	7
Média iter. DSC: 8,86			**Média iter. CPLEX:** 5,93	

Como era esperado, a estrutura de dados neste caso não influenciou no número de iterações, pois no presente trabalho, apenas uma regra de escolha da entrada de uma nova variável na base em cada iteração foi utilizada, conhecida como regra usual de Dantzig, ou seja a escolha é feita numa direção simplex que possui o coeficiente mais negativo, Bazaraa(2010) [3]. Variações na regra de Dantzig, tais como a regra de Dantzig Normalizada (*Steepest Edge Rule*), podem afetar diretamente o número de iterações do método, como apresentado em Sousa [40].

Também pode ser notado na Tabela 6.5, que a relação entre o número de iterações e a dimensão dos problemas foi muito variada, mas sempre inferior ao número de linhas e próximo do número de colunas.

O algoritmo Dual Simplex do *solver* CPLEX obteve uma melhor média que o DSC em todas as instâncias em relação ao número de iterações, com exceção da instância "inst13_prob_mistura". Em relação ao tempo, o CPLEX 12.8 também obteve valores inferiores também a 10^{-3} segundos, desta forma a única informação apresentada diz respeito

ao número de iterações.

As instâncias do problema da fábrica de ração são pequenas, mas importantes sobre o ponto de vista de representarem problemas na forma geral canalizada e que são de fato reais. Vale ressaltar que, mesmo uma implementação comercial do Dual Simplex pode realizar mais iterações em um determinado problema, como aconteceu com a instância 13.

6.2.2 Problemas de Programação da Produção

A Tabela 6.6 mostra os dados de solução das instâncias pequenas do problema de programação da produção no DSC usando a forma Revisada em cada uma das estruturas de dados. Vale ressaltar que o número de iterações foi o mesmo independente da estrutura utilizada.

Tabela 6.6: Execução DSC Revisado - Instâncias Pequenas.

Instância	**IT**	**Index**	**LE**	**CSC/CSR**
		Tempo Sol. (s)	**Tempo Sol. (s)**	**Tempo Sol. (s)**
inst1_prod_60x120	64	0,004	0,012	0,003
inst2_prod_110x230	118	0,028	0,083	0,019
inst3_prod_160x340	168	0,081	0,244	0,056
inst4_prod_210x450	222	0,189	0,568	0,129
inst5_prod_260x560	282	0,386	1,124	0,247
inst6_prod_310x670	353	0,687	2,113	0,445
inst7_prod_360x780	392	1,036	3,061	0,668
inst8_prod_410x890	443	1,634	4,591	0,957
inst9_prod_460x1000	509	2,300	6,455	1,406
Média		**0,705**	**2,028**	**0,437**

Observa-se que, em média, o tempo de solução da implementação utilizando Listas Encadeadas (LE) foi quase 3 vezes mais lenta que a versão Indexada e quase 5 vezes em relação abordagem com vetores compactos CSC/CSR. Isto se deve ao fato da maneira com que as operações são realizadas durante as iterações do método, tais como o acesso aos elementos da matriz de restrições do problema e a atualização da matriz básica.

O acesso aos elementos da lista atua como um fator de influência direta no desempenho, visto que não há um índice indicando a posição exata do elemento procurado, sendo assim o acesso no pior caso se dá em uma complexidade assintótica de $O(n)$. A remoção na lista também influencia, pois antes de executar a operação, é necessário identificar o elemento a ser removido no processo de atualização da base, o que leva ao caso anterior.

A estrutura de dados Indexada, apesar de armazenar elementos nulos e realizar

operações com eles, obteve vantagem pela sua característica de se ter um acesso constante (*O(1)*). Desta forma, os cálculos realizados durante as iterações tiram proveito desta particularidade.

Observe que o número de iterações é muito próximo da quantidade de linhas e bem menor que o número de colunas para cada problema, ou seja, um bom resultado para o DSC.

A Figura 6.3 ilustra o desempenho em termos dos tempos computacionais para as 3 estruturas.

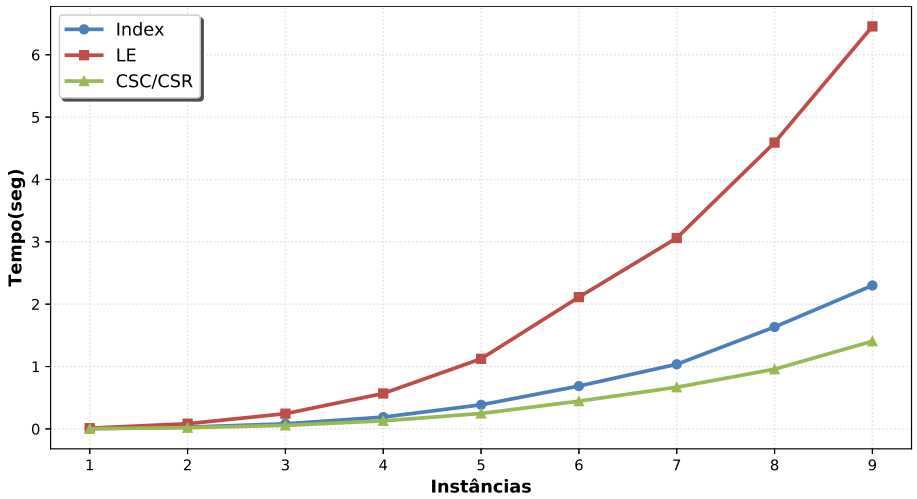

Figura 6.3: Gráfico Tempo de Execução DSC Revisado - Problemas de Programação da Produção - Instâncias Pequenas.

Pelo gráfico, torna-se mais evidente as diferenças apresentadas nos tempos para as 3 estruturas de dados, com destaque para o desempenho da estrutura CSC/CSR.

A estrutura de dados utilizando vetores compactos CSC/CSR foi o destaque em relação às demais, sendo em média, 4,6 e 1,6 vezes mais rápida que a LE e Indexada, respectivamente.

Na Tabela 6.7, encontram-se os resultados para as mesmas instâncias pequenas do problema de programação da produção no DSC, mas agora utilizando a FPI para atualização da base. Para as 3 estruturas de dados, o número de iterações realizadas pelo método foi o mesmo.

Tabela 6.7: Execução DSC FPI - Instâncias Pequenas.

Instância	IT	Index	LE	CSC/CSR
		Tempo Sol. (s)	Tempo Sol. (s)	Tempo Sol. (s)
inst1_prod_60x120	64	0,001	0,001	0,001
inst2_prod_110x230	118	0,005	0,001	0,001
inst3_prod_160x340	168	0,013	0,002	0,002
inst4_prod_210x450	222	0,028	0,003	0,003
inst5_prod_260x560	282	0,054	0,005	0,005
inst6_prod_310x670	353	0,096	0,008	0,006
inst7_prod_360x780	392	0,143	0,010	0,008
inst8_prod_410x890	443	0,212	0,011	0,010
inst9_prod_460x1000	509	0,314	0,016	0,012
Média		**0,096**	**0,006**	**0,005**

Como pode ser visto, a abordagem utilizando a FPI como forma de atualização da base foi destacadamente mais eficiente, ou seja, fez reduzir drasticamente o tempo de solução quando comparada com a abordagem Revisada. Isto ocorre devido a forma de como a matriz básica é atualizada e armazenada em cada iteração e a maneira pela qual os elementos nas estruturas CSC/CSR são acessados. Como descrito no Capítulo 4, a matriz básica é armazenada através de matrizes de transformações elementares compostas por vetores esparsos, sendo assim todas as operações são beneficiadas por esta abordagem.

Um fato que pode ser observado é que em todas as estruturas de dados os tempos foram inferiores a 0,4 segundos, evidenciando assim um bom desempenho quando se utiliza a FPI. A Figura 6.4 mostra os resultados da Tabela 6.7 evidenciando as diferenças.

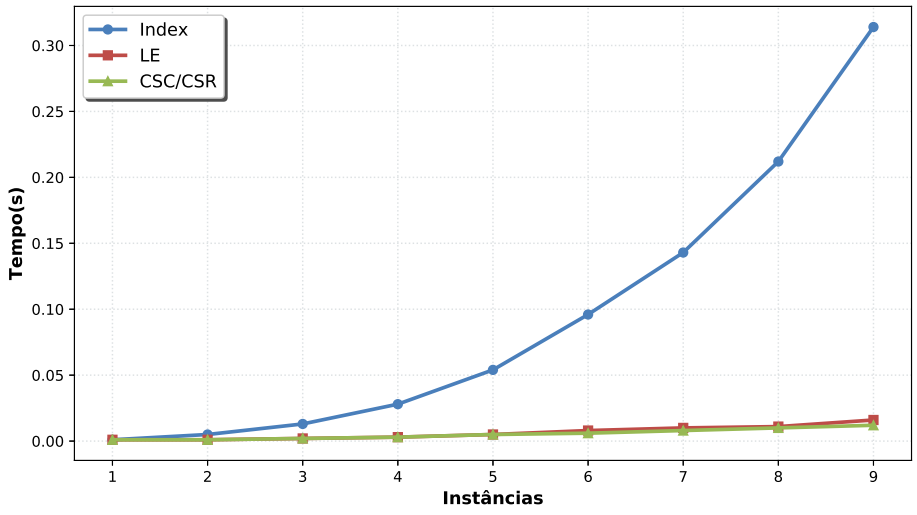

Figura 6.4: Gráfico Tempo de Execução DSC FPI - Problemas de Programação da Produção - Instâncias Pequenas.

Pelo gráfico, pode-se perceber de maneira mais explícita o comportamento da implementação FPI nestas instâncias pequenas.

Note também que, diferentemente da implementação na forma Revisada, a estrutura de dados utilizando Listas Encadeadas (LE) obteve, em média, praticamente o mesmo tempo de solução da estrutura usando vetores compactos CSC/CSR.

Agora, na Tabela 6.8 são apresentados os resultados para as instâncias de tamanho médio com o método DSC utilizando a forma Revisada.

Tabela 6.8: Execução DSC Revisado - Instâncias Médias.

Instância	**IT**	**Index**	**LE**	**CSC/CSR**
		Tempo Sol. (s)	**Tempo Sol. (s)**	**Tempo Sol. (s)**
inst10_prod_510x1110	559	3,241	6,68	1,867
inst11_prod_800x1580	948	10,405	33,135	6,459
inst12_prod_900x1660	1078	13,716	39,043	7,932
inst13_prod_1050x2090	1250	25,043	75,287	14,94
inst14_prod_1200x2210	1456	32,685	107,129	18,857
inst15_prod_1300x2600	1540	45,788	140.685	28,138
inst16_prod_1550x3110	1868	92,867	243.737	49,282
inst17_prod_1800x3620	2174	124.403	403,605	76,805
inst18_prod_2050x4130	2433	183,404	535,488	111,758
inst19_prod_2200x4220	2611	205,417	658,168	124,554
inst20_prod_2300x4640	2741	257,128	802,855	179,253
Média		**90,372**	**276,892**	**56,349**

Com relação aos tempos, nota-se um aumento significativo em comparação com os problemas pequenos, o que é natural devido ao aumento na dimensão de cada problema ($m \times n$). Novamente observa-se os mesmos comportamentos em relação aos tempos de solução para cada estrutura de dados utilizada, sendo CSC/CSR a melhor abordagem e LE consumindo muito tempo, em média, quase 5 vezes mais que CSC/CSR e aproximadamente 3 vezes mais que Indexada.

É importante observar que, no geral o número de iterações foi pouco superior ao número de linhas do problema e bem menor que o número de colunas. Neste caso, o DSC efetuou um número baixo de iterações para encontrar a solução ótima em função da dimensão de cada problema.

A Tabela 6.9 mostra os resultados de execução das instâncias de tamanho médio, agora para o método DSC utilizando a forma FPI.

Tabela 6.9: Execução DSC FPI - Instâncias Médias.

Instância	**IT**	**Index**	**LE**	**CSC/CSR**
		Tempo Sol. (s)	**Tempo Sol. (s)**	**Tempo Sol. (s)**
inst10_prod_510x1110	559	0,436	0,020	0,015
inst11_prod_800x1580	948	1,815	0,044	0,036
inst12_prod_900x1660	1078	2,431	0,056	0,041
inst13_prod_1050x2090	1250	4,115	0,084	0,062
inst14_prod_1200x2210	1456	5,801	0,097	0,077
inst15_prod_1300x2600	1540	7,776	0,124	0,094
inst16_prod_1550x3110	1868	13,400	0,194	0,139
inst17_prod_1800x3620	2174	21,556	0,278	0,205
inst18_prod_2050x4130	2433	30,726	0,341	0,245
inst19_prod_2200x4220	2611	36,901	0,361	0,262
inst20_prod_2300x4640	2741	43,222	0,424	0,315
Média		**15,289**	**0,166**	**0,136**

Como pode ser observado, não há variações no número de iterações, e no geral a FPI faz reduzir significativamente o tempo. A estrutura de dados com LE obteve um desempenho próximo a estrutura de dados CSC/CSR, diferentemente dos problemas pequenos. A estrutura de dados Indexada levou muito mais tempo para encontrar a solução do que as outras duas (LE e CSC/CSR).

A Tabela 6.10 mostra os resultados das instâncias de tamanho grande para a implementação do DSC utilizando a forma Revisada. Ressalta-se que não são mostrados os resultados para as instâncias de número 33 até 36, pois estas instâncias não foram executadas. O motivo da não execução destas instâncias, reside no fato de que tomando por base o tempo de solução com LE da instância inst28_prod_5100x10200 juntamente com a instância inst32_prod_8200x16280, existe uma relação de aproximadamente 4,4 vezes neste tempo. Neste caso o tempo de execução sai de 2,5h para 11,3h, sendo importante ressaltar que a relação entre o número de variáveis e número de restrições de ambas instâncias citadas seguem uma proporção de aproximadamente 1,6.

Agora, considerando a instância inst32_prod_8200x16280 e seu tempo de solução com LE e a inst36_prod_13260x26360 e levando em consideração que ambas também mantém a mesma proporção em relação ao número de variáveis e restrições (aproximdamente 1,6), estima-se que o tempo necessário para resolver a instância inst36_prod_13260x26360 seria de aproximadamente 49,7h. Portanto, este tempo foi considerado demasiadamente alto para efeitos de comparação.

Tabela 6.10: Execução DSC Revisado - Instâncias Grandes.

Instância	IT	Index	LE	CSC/CSR
		Tempo Sol. (s)	Tempo Sol. (s)	Tempo Sol. (s)
inst21_prod_2550x5150	3038	341,970	1124,062	221,660
inst22_prod_3150x6190	3800	629,354	1990,594	400,135
inst23_prod_3600x7170	4369	932,681	3155,559	622,804
inst24_prod_3720x7380	4474	1096,052	3408,800	692,058
inst25_prod_3900x7700	4781	1177,211	4144,191	793,856
inst26_prod_4100x8180	5046	1463,371	4973,554	927,176
inst27_prod_4600x9190	5608	1981,006	6937,431	1300,566
inst28_prod_5100x10200	6161	2763,343	9299,890	1806,131
inst29_prod_6120x12220	7364	4569,134	14860,162	3041,056
inst30_prod_6900x13740	8481	6775,075	22952,597	4426,140
inst31_prod_7200x14270	8912	7921,783	26817,645	4867,519
inst32_prod_8200x16280	10009	11606,152	40839,257	7322,193
Média		**3438,090**	**11708,645**	**2201,775**

Pode-se notar claramente o aumento no tempo de solução devido ao aumento no tamanho das instâncias. A implementação utilizando LE, novamente obteve um desempenho muito ruim em relação as demais para a forma Revisada. O número de iterações foi em média $22,3\%$ superior ao número de linhas dos problemas, mas ainda assim, configurando um bom desempenho para o DSC. A estrutura de dados CSC/CSR foi novamente bem superior as demais, sendo em média cerca de 36% mais rápida em relação a estrutura Indexada e mais 80% em relação a implementação com LE.

Na Tabela 6.11 apresenta-se os resultados das instâncias de tamanho grande para a implementação do DSC utilizando a FPI.

Tabela 6.11: Execução DSC FPI - Instâncias Grandes.

Instância	IT	Index	LE	CSC/CSR
		Tempo Sol. (s)	Tempo Sol. (s)	Tempo Sol. (s)
inst21_prod_2550x5150	3038	58,961	0,534	0,440
inst22_prod_3150x6190	3800	109,214	0,775	0,642
inst23_prod_3600x7170	4369	166,999	1,063	0,897
inst24_prod_3720x7380	4474	181,192	1,269	0,917
inst25_prod_3900x7700	4781	220,932	1,255	1,040
inst26_prod_4100x8180	5046	252,168	1,501	1,221
inst27_prod_4600x9190	5608	350,710	1,881	1,525
inst28_prod_5100x10200	6161	472,147	2,298	1,853
inst29_prod_6120x12220	7366	811,977	3,471	2,624
inst30_prod_6900x13740	8490	1236,620	4,908	3,186
inst31_prod_7200x14270	8916	1498,936	5,376	3,539
inst32_prod_8200x16280	10003	2199,233	6,923	4,390
inst33_prod_9200x18290	11247	2973,187	9,412	5,661
inst34_prod_10200x20300	12449	3908,363	11,718	6,997
inst35_prod_11235x22365	13741	5281,832	14,576	8,910
inst36_prod_13260x26360	16534	8676,396	25,812	14,019
Média	7876,437	**1774,929**	**5,798**	**3,616**

Novamente, destaca-se o tempo de solução bem inferior ao se utilizar as estruturas CSC/CSR e LE, com desvantagem para esta última e desempenho muito ruim quando se utiliza a estrutura Indexada. Ou seja, a forma de atualização da base com a abordagem FPI e estruturas de dados apropriadas (CSC/CSR e LE) faz reduzir significativamente o tempo de solução para as instâncias grandes desta classe de problema abordado.

Com o objetivo de verificar a diferença nas médias dos tempos de solução das instâncias do problema da produção usando ambas as formas de atualização da base, as Figuras 6.5 e 6.6 sumarizam as médias das Tabelas 6.6, 6.8 e 6.10 para a forma Revisada e das Tabelas 6.7, 6.9 e 6.11 para a FPI. Convém destacar que para o cálculo da média das instâncias grandes referentes a Tabela 6.11 foram consideradas as mesmas instâncias da Tabela 6.10, ou seja, instâncias 21 até 32.

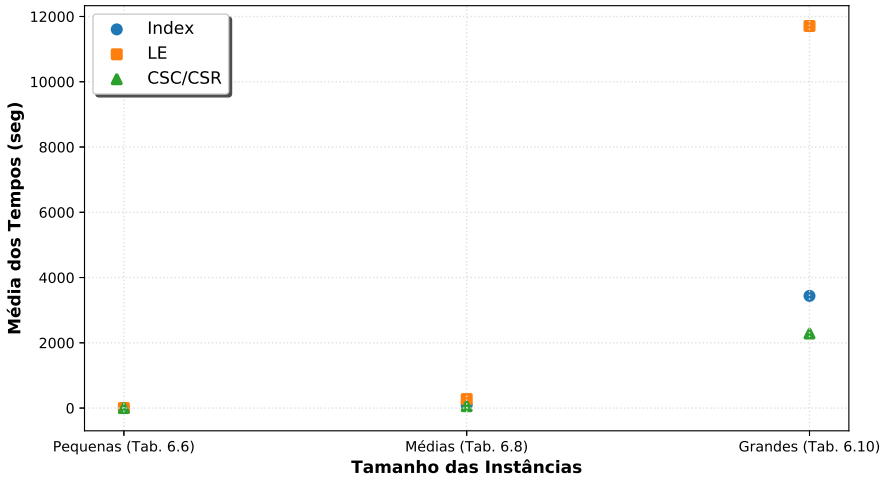

Figura 6.5: Gráfico com média dos tempos DSC Revisado - Problemas de Programação da Produção

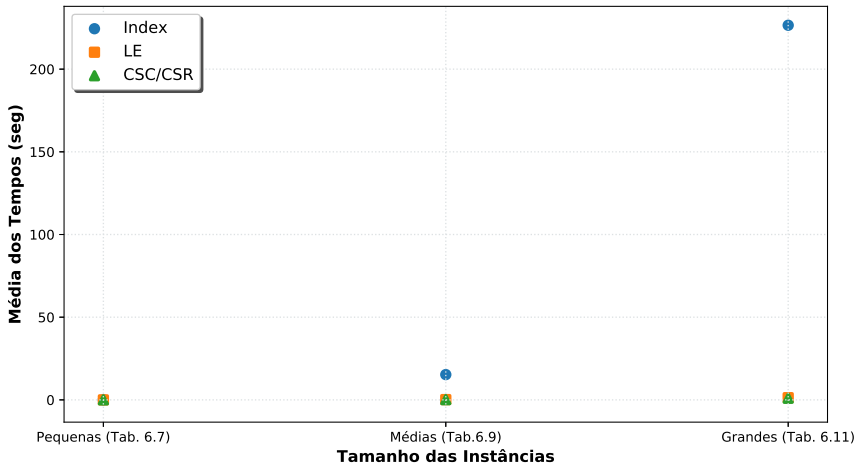

Figura 6.6: Gráfico com média dos tempos DSC FPI - Problemas de Programação da Produção

Como pode ser notado, a estrutura de dados com LE não se mostrou muito eficiente com a forma Revisada para resolver problemas médios e grandes, contudo para a FPI o desempenho desta estrutura foi praticamente o mesmo da estrutura CSC/CSR.

Agora, para efeito de comparação, a Tabela 6.12 mostra os resultados obtidos no Dual Simplex presente no CPLEX 12.8 para os problemas grandes de produção.

Tabela 6.12: Execução CPLEX 12.8 - Instâncias Grandes.

Instância	**IT**	**Tempo Sol. (s)**
inst21_prod_2550x5150	2727	0,060
inst22_prod_3150x6190	3402	0,060
inst23_prod_3600x7170	3970	0,060
inst24_prod_3720x7380	4057	0,060
inst25_prod_3900x7700	4433	0,080
inst26_prod_4100x8180	4668	0,080
inst27_prod_4600x9190	5225	0,080
inst28_prod_5100x10200	5718	0,080
inst29_prod_6120x12220	6867	0,090
inst30_prod_6900x13740	7919	0,090
inst31_prod_7200x14270	8349	0,090
inst32_prod_8200x16280	9373	0,110
inst33_prod_9200x18290	10542	0,130
inst34_prod_10200x20300	11768	0,140
inst35_prod_11235x22365	13018	0,160
inst36_prod_13260x26360	15660	0,190
Média	7356	**0,0975**

Como pode ser visto, a implementação do algoritmo Dual Simplex contida no *solver* CPLEX 12.8 encontrou a solução ótima para todos os problemas em um tempo muito menor quando comparado ao DSC utilizando a FPI e a estrutura de dados CSC/CSR.

Ressalta-se que o *solver* CPLEX 12.8 é um *software* com mais de 3 décadas de existência, tendo como resultado uma constante evolução ao longo dos anos principalmente em tempo computacional, como pode ser visto em Bixby [4]. Cabe destacar que o DSC com as 3 estruturas de dados realizou mais iterações que o Dual Simplex do CPLEX 12.8, mas não foi uma grande superioridade para o CPLEX 12.8, visto que precisou, em média, cerca de $6,6\%$ mais iterações que a versão presente no *solver*.

6.2.3 Problemas do Repositório da NetLib

A implementação do DSC utilizando a forma Revisada resolveu mais instâncias que a implementação utilizando a FPI para as instâncias da NetLib, conseguindo resolver 23 pequenas e 9 médias. Provavelmente, isto se deve ao fato que na implementação FPI não foi utilizada a refatoração da base após um certo número de iterações. Este processo faz com que se tenha uma melhor estabilidade numérica durante a execução do método simplex, Munari [25].

6.2.3.1 DSC com a Forma Revisada

Na Tabela 6.13, encontram-se os resultados para os problemas da NetLib classificados como pequenos resolvidos pelo DSC usando a forma Revisada.

Tabela 6.13: Execução DSC Revisado - Instâncias Pequenas.

Instancia	**IT**	**Index**	**LE**	**CSC/CSR**
		Tempo Sol. (s)	**Tempo Sol. (s)**	**Tempo Sol. (s)**
AFIRO	26	0,001	0,001	0,000
KB2	187	0,003	0,015	0,003
SC50B	51	0,001	0,005	0,001
SC50A	48	0,001	0,005	0,001
SHARE2B	324	0,011	0,162	0,012
ADLITTLE	147	0,006	0,079	0,011
SC105	110	0,007	0,097	0,008
SCAGR7	349	0,032	0,792	0,044
RECIPELP	45	0,007	0,016	0,004
VTP-BASE	639	0,119	4,791	0,121
SC205	206	0,044	1,575	0,056
SHARE1B	693	0,148	9,355	0,209
E226	948	0,330	36,563	0,500
AGG2	306	0,151	0,379	0,078
AGG3	334	0,159	0,563	0,088
SCORPION	427	0,256	8,151	0,211
SCTAP1	535	0,516	41,365	0,710
SCAGR25	1161	1,255	236,670	1,522
DEGEN2	2735	3,307	1507,455	6,559
FINNIS	584	0,946	57,861	0,805
ETAMACRO	1211	2,123	893,702	3,732
SCSD1	160	0,371	12,471	0,340
Média	510,272	**0,445**	**127,821**	**0,682**

É importante notar que algumas instâncias (AFIRO, SC50A, SC50B, SC105, RECIPELP, SC205, AGG2, AGG3, SCORPION e FINNIS), o DSC realizou poucas iterações para

encontrar o valor ótimo em relação ao número de linhas destes problemas. Nos demais problemas, o número de iterações foi em média $3,5m$, sendo m o número de linhas. Isto se deve ao fato de serem problemas numericamente difíceis de serem resolvidos e principalmente por serem altamente degenerados, como é o caso do DEGEN2 que por consequência levou maior tempo para ser resolvido.

No geral, os tempos de execução foram baixos tanto para as estruturas Indexada e CSC/CSR em todos os problemas, com uma pequena vantagem, neste caso, para a estrutura Indexada. Como pode ser visto, o DSC usando a estrutura de LE levou muito tempo para resolver alguns problemas, em especial as instâncias DEGEN2, ETAMACRO e SCAGR25. Pan [35] relata que para alguns problemas, de 10% a 57% dos passos para se encontrar a solução são degenerados. Segundo o autor, os problemas KB2 e DEGEN2 possuem, respectivamente, 35,37% e 56,74% dos passos degenerados.

Na Tabela 6.14 encontram-se os resultados obtidos para as mesmas instâncias com o Dual Simplex do CPLEX 12.8.

Tabela 6.14: Execução CPLEX 12.8 - Instâncias Pequenas.

Instância	**IT**	**Tempo Sol. (s)**
AFIRO	8	0,020
KB2	43	0,010
SC50B	38	0,000
SC50A	34	0,000
SHARE2B	112	0,000
ADLITTLE	119	0,020
SC105	62	0,020
SCAGR7	175	0,020
RECIPELP	45	0,010
VTP-BASE	193	0,020
SC205	150	0,020
SHARE1B	201	0,010
E226	510	0,020
AGG2	131	0,020
AGG3	135	0,020
SCORPION	300	0,020
SCTAP1	207	0,020
SCAGR25	505	0,030
DEGEN2	658	0,050
FINNIS	436	0,010
ETAMACRO	894	0,030
SCSD1	110	0,000
Média	230,272	**0,016**

A implementação do Dual Simplex presente no CPLEX 12.8 encontrou os valores

ótimos para todos os problemas com um número de iterações muito menor que o DSC, com exceção do problema RECIPELP, pois para este o número de iterações foi o mesmo.

A média das iterações em relação ao número de linhas do problema, foi de 1,04m, constituindo assim um bom desempenho em relação ao DSC. É importante relatar que o CPLEX possui técnicas avançadas para o tratamento de problemas altamente degenerados o que também contribui para esta vantagem. Em relação ao tempo de solução, estes foram bem menores que o DSC, mas o tempo médio do DSC com estruturas Indexada e CSC/CSR foi de aproximadamente 0,5 segundos.

A seguir, na Tabela 6.15 estão os resultados do DSC na Forma Revisada para as instâncias médias. Cabe ressaltar que a instância SHIP12L possui 5427 variáveis, o que a classificaria como instância grande pelo padrão adotado, mas esta foi incluída junto as médias por ser a única com número de variáveis acima de 5000 que o método conseguiu encontrar a solução.

Tabela 6.15: Execução DSC Revisado - Instâncias Médias.

Instancia	**IT**	**Index**	**LE**	**CSC/CSR**
		Tempo Sol. (s)	**Tempo Sol. (s)**	**Tempo Sol. (s)**
STANDATA	79	0,408	1,051	0,206
STANDMPS	276	1,527	8,388	0,820
SCRS8	1062	5,884	878,942	5,665
SHIP04S	402	3,586	11,407	2,019
SHELL	735	9,478	67,556	5,843
SCTAP2	989	16,243	3671,947	13,519
SHIP04L	396	7,407	22,880	4,024
SHIP12S	1120	34,784	128,223	21,516
SHIP12L	1186	142,160	527,601	86,425
Média	693,888	**24,608**	**590,888**	**15,559**

Nota-se um bom desempenho do DSC em relação ao número de iterações realizadas para encontrar as soluções ótimas destes problemas. Sendo em média 1,08m em relação ao número de linhas do problema. O destaque vai para a solução do problema STANDATA, pois a solução foi encontrada com um número de iterações cerca de 4,5 vezes menor que seu número de linhas. O número de iterações realizadas para o problema STANDMPS também foi bem menor que o número de linhas deste.

No geral, os tempos de solução aumentaram devido ao crescimento dos problemas, principalmente em relação ao número de variáveis. Este substancial aumento do número de variáveis não influenciou diretamente no número de iterações, visto que a relação entre as iterações e o número de linhas dos problemas foi de 1,02.

Novamente evidencia-se a vantagem da estrutura CSC/CSR, sendo esta em média 37,9 e 1,5 vezes mais rápida que LE e Indexada, respectivamente.

Os resultados obtidos pelo Dual Simplex do CPLEX 12.8 para os problemas anteriores encontram-se na Tabela 6.16.

Tabela 6.16: Execução CPLEX 12.8 - Instâncias Médias.

Instância	**IT**	**Tempo Sol. (s)**
STANDATA	185	0,020
STANDMPS	216	0,050
SCRS8	622	0,020
SHIP04S	243	0,020
SHELL	536	0,030
SCTAP2	495	0,020
SHIP04L	291	0,030
SHIP12S	467	0,000
SHIP12L	785	0,050
Média	426,667	**0,026**

Pode-se perceber novamente, que o CPLEX 12.8 fez, para a maioria dos problemas, um número de iterações bem menor que o DSC para encontrar a solução ótima e com um tempo extremamente baixo. No entanto, para o problema STANDATA, o CPLEX 12.8 realizou 2,34 vezes mais iterações que o DSC, configurando assim um bom desempenho para o DSC. O número de iterações foram, em média, cerca de 0,71m, sendo m o número de linhas dos problemas.

6.2.3.2 DSC com a Forma Produto da Inversa

Como já foi relatado, o número de instâncias da NetLib solucionadas pela implementação do método DSC utilizando a FPI foi menor em relação a Forma Revisada, no caso 6 pequenas e 7 médias. A seguir, os resultados obtidos para os problemas classificados como pequenos são sumarizados na Tabela 6.17.

Tabela 6.17: Execução DSC FPI - Instâncias Pequenas.

Instancia	IT	Index	LE	CSC/CSR
		Tempo Sol. (s)	Tempo Sol. (s)	Tempo Sol. (s)
AFIRO	27	0,000	0,001	0,000
KB2	122	0,001	0,003	0,000
SCAGR7	346	0,020	0,023	0,012
RECIPELP	45	0,001	0,000	0,001
SCAGR25	1161	0,764	0,577	0,210
DEGEN2	2624	4,131	10,536	5,290
Média		**0,820**	**1,857**	**0,919**

Como pode ser visto, o número de iterações com FPI foi ligeiramente diferente em relação aos mesmos problemas com a forma Revisada conforme a Tabela 6.13, sendo menores para alguns problemas (AFIRO, KB2, SCAGR7 e DEGEN2). Em relação aos tempos de solução nota-se também uma melhora significativa, principalmente com a estrutura usando LE, principalmente para os problemas SCAGR25 e DEGEN2.

Novamente a estrutura de dados CSC/CSR se destaca por seu desempenho frente as demais, tendo uma melhora geral em relação aos mesmos problemas comparados a abordagem Revisada. A estrutura Indexada também obteve uma melhora, com exceção do problema DEGEN2.

Os tempos de solução dos 4 primeiros problemas na Tabela 6.17 com estrutura CSC/CSR foram melhores que o Dual Simplex do CLPEX 12.8 quando comparados com os resultados da Tabela 6.14 para estes mesmos problemas. Isso evidencia mais uma vez a vantagem da utilização deste tipo de estrutura de dados juntamente com a abordagem FPI.

Por fim, na Tabela 6.18 encontram-se os resultados para os problemas classificados como médios ($n \leq 5000$) da NetLib resolvidos pelo `DSC` na abordagem FPI.

Tabela 6.18: Execução DSC FPI - Instâncias Médias.

Instância	IT	Index	LE	CSC/CSR
		Tempo Sol. (s)	Tempo Sol. (s)	Tempo Sol. (s)
STANDATA	81	0,041	0,005	0,003
SHIP04S	402	0,305	0,039	0,021
SHELL	875	1,170	0,953	0,300
SHIP04L	396	0,483	0,061	0,031
SHIP12S	1120	5,213	0,472	0,171
SHIP12L	1187	11,212	1,612	0,396
Média	676,833	**3,070**	**0,523**	**0,153**

Mais uma vez o DSC com a FPI fez reduzir drasticamente os tempos de solução quando comparada a forma Revisada (Tabela 6.15) para os mesmos problemas. A estrutura Indexada e principalmente para as estruturas LE e CSC/CSR em relação aos problemas SHIP12S e SHIP12L tiveram tempos de solução extremamente baixos quando comparados a forma Revisada, pois estes foram resolvidos com tempos de aproximadamente 1s, sendo que anteriormente este tempo era de dezenas de segundos. Em geral a estrutura CSC/CSR se mostrou novamente ser a mais eficiente, seguida pela LE, indicando assim que a LE com a FPI também configura-se uma boa escolha de implementação.

Em relação ao número de iterações, também nota-se uma ligeira diferença em relação aos mesmos problemas quando comparados a solução com a Forma Revisada. Sendo que, para o problema `SHELL`, a forma de atualização usando FPI realizou 140 iterações a mais que a forma Revisada, mas com um tempo de resolução muito melhor. Nos demais problemas, as diferenças foram muito pequenas.

Pode-se perceber novamente, ao se comparar os resultados da Tabela 6.18 com a Tabela 6.15 que o CPLEX 12.8 fez, para a maioria dos problemas, um número de iterações bem menor que o DSC para encontrar a solução ótima.

Destaca-se que para alguns problemas, os tempos de solução do `DSC` não foram ruins quando comparados ao CPLEX 12.8, pois deve-se considerar fato que o DSC realizou mais iterações.

É importante ressaltar que as soluções encontradas pelo DSC para os problemas do repositório da NetLib aqui analisados possuem 11 dígitos de precisão, sendo esta, a mesma precisão numérica apresentada no trabalho de Koch [31], com exceção do problema SCAGR25 que diferiu dos resultados de Koch na décima casa decimal para a abordagem Revisada e `SCAGR7` na oitava casa decimal para a abordagem FPI. Estes resultados sugerem um bom desempenho e a robustez do método DSC para solução destas instâncias, pois elas representam problemas reais e que são reconhecidamente difíceis de serem resolvidos e tratados numericamente, como ressaltado em Bixby [7].

6.2.4 Tempo gasto nas principais rotinas do DSC para algumas instâncias grandes

Nesta seção, para as 8 primeiras instâncias classificadas como grandes do problema de programação da produção, e resolvidas pelo DSC utilizando as estruturas de dados Indexada e CSC/CSR e as duas formas de atualização (Revisada e FPI), são apresentados os tempos por iteração e a porcentagem de tempo utilizada nas principais rotinas do método DSC em relação ao tempo total de solução. As rotinas analisadas de acordo com o Algoritmo 1 foram: Teste ótimo, Cálculo da direção dual simplex, Tamanho do passo e Atualização.

Na Tabela 6.19 encontram-se os dados com a forma de atualização Revisada.

Tabela 6.19: Tempo por rotinas - DSC Revisado - Estrutura Indexada - Instâncias Grandes

Instância	**IT**	**Tempo Sol.(s)**	**Tempo Iter.(s)**	**Teste ótimo(%)**	**Dir. Dual(%)**	**Tam. Passo(%)**	**Atual.(%)**
inst21_prod_2550x5150	3038	341,970	0,113	7,07	25,52	0,02	67,39
inst22_prod_3150x6190	3800	629,354	0,166	7,31	25,24	0,02	67,43
inst23_prod_3600x7170	4369	932,681	0,213	7,24	25,23	0,01	67,52
inst24_prod_3720x7380	4474	1096,052	0,245	6,92	24,03	0,02	69,03
inst25_prod_3900x7700	4781	1177,211	0,246	7,27	25,06	0,01	67,66
inst26_prod_4100x8180	5046	1463,371	0,290	7,37	24,96	0,02	67,65
inst27_prod_4600x9190	5608	1981,006	0,353	7,12	25,13	0,02	67,73
inst28_prod_5100x10200	6161	2763,343	0,449	7,29	25,29	0,02	67,40
Média:	4659,625	1298,124	0,259	7,20	25,06	0,02	67,72

Como pode ser visto, a rotinas onde o DSC consumiu mais tempo foram: o processo de atualização seguida pelo cálculo da direção dual simplex e por último, a realização do teste ótimo. Cabe ressaltar aqui, que na rotina de atualização, além da base, também são atualizadas a solução dual e primal, o que contribui para o maior tempo consumido. Considerando o Passo 4 (P4) do Algoritmo 1 para a forma Revisada, temos que a atualização dos índices e solução dual são procedimentos rápidos, ao passo que a atualização da base e cálculo da solução primal consomem mais tempo devido ao número de operações envolvidas, sendo assim esta parte do algoritmo foi desenvolvido justamente para reduzir as operações nestas atualizações.

O cálculo da direção dual simplex representou, aproximadamente, 25% do tempo total devido a necessidade de solução do cálculo do sistema linear para a obtenção de suas componentes a cada iteração. Por fim, na rotina de teste ótimo, a análise do maior custo relativo entre todas as variáveis não básicas e a verificação de possíveis violações nos limitantes fizeram com que este processo também utilizasse parte considerável do tempo.

Nota-se também que o tempo por iteração aumenta gradativamente a medida que cresce a dimensão dos problemas.

Agora, na Tabela 6.20 encontram-se sumarizados os cálculos dos tempos médios das rotinas do DSC utilizando a FPI para as mesmas instâncias.

Tabela 6.20: Tempo por rotinas - DSC FPI - Estrutura Indexada - Instâncias Grandes

Instância	**IT**	**Tempo Sol.(s)**	**Tempo Iter.(s)**	**Teste ótimo(%)**	**Dir. Dual(%)**	**Tam. Passo(%)**	**Atual.(%)**
inst21_prod_2550x5150	3038	59,098	0,019	42,59	0,12	0,12	57,17
inst22_prod_3150x6190	3800	109,933	0,029	42,19	0,18	0,11	57,52
inst23_prod_3600x7170	4369	159,753	0,037	42,16	0,08	0,07	57,69
inst24_prod_3720x7380	4474	179,253	0,040	42,96	0,17	0,10	56,77
inst25_prod_3900x7700	4779	220,932	0,046	41,51	0,07	0,07	58,35
inst26_prod_4100x8180	5046	247,074	0,049	42,24	0,15	0,11	57,50
inst27_prod_4600x9190	5608	338,026	0,060	42,26	0,15	0,07	57,52
inst28_prod_5100x10200	6161	462,136	0,075	42,15	0,15	0,10	57,60
Média:	4659,375	222,026	0,048	42,25	0,13	0,09	57,51

Nota-se que na FPI, a rotina de atualização ainda é a que consome mais tempo. Mas agora, diferentemente da abordagem revisada, a porcentagem média do tempo gasto no cálculo da direção dual é muito menor, chegando a ser menor que 1%. Este resultado está diretamente relacionado a maneira com que a direção dual é obtida com a FPI. Como descrito no Capítulo 4, este cálculo é realizado por meio da rotina FTRAN. Em segundo lugar entre as maiores médias, também encontra-se o teste ótimo, evidenciando que seu processo de cálculo dos custos e verificação de violações constituem um considerável esforço computacional para esta forma de atualização.

Para melhor visualização das diferenças nas porcentagens médias mostradas nas Tabelas 6.19 e 6.20, o gráfico da Figura 6.7 é apresentado a seguir.

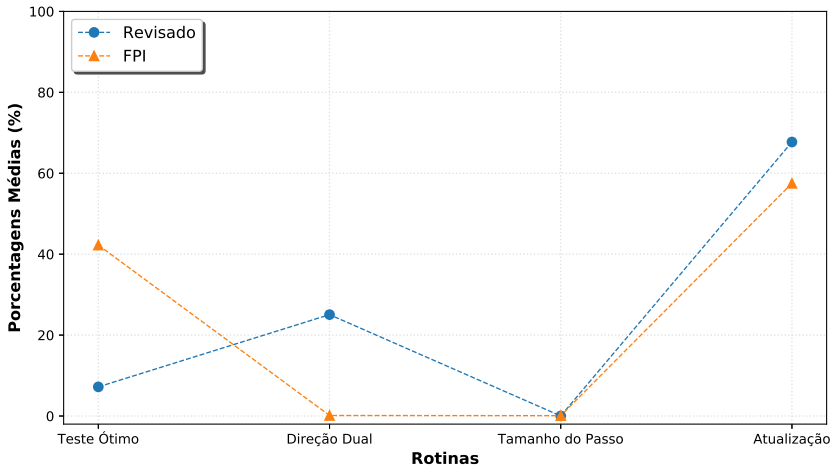

Figura 6.7: Porcentagem média das rotinas - DSC Revisado e FPI - Estrutura Indexada

Agora na Tabela 6.21 são apresentados os dados referentes ao consumo de tempo das rotinas com a forma Revisada utilizando a estrutura de dados CSC/CSR.

Tabela 6.21: Tempo por rotinas - DSC Revisado - Estrutura CSC - Instâncias Grandes

Instância	**IT**	**Tempo Sol.(s)**	**Tempo Iter.(s)**	**Teste ótimo(%)**	**Dir. Dual(%)**	**Tam. Passo(%)**	**Atual.(%)**
inst21_prod_2550x5150	3038	221,66	0,072	0,06	0,09	0,021	99,82
inst22_prod_3150x6190	3800	400,135	0,105	0,06	0,09	0,02	99,83
inst23_prod_3600x7170	4369	622,804	0,142	0,05	0,08	0,02	99,84
inst24_prod_3720x7380	4474	692,058	0,154	0,05	0,08	0,01	99,85
inst25_prod_3900x7700	4779	793,856	0,166	0,05	0,07	0,01	99,86
inst26_prod_4100x8180	5046	927,176	0,183	0,05	0,07	0,02	99,86
inst27_prod_4600x9190	5608	1300,566	0,231	0,05	0,82	0,01	99,87
inst28_prod_5100x10200	6161	1806,131	0,293	0,04	1,07	0,01	99,89
Média:	4659,375	845,548	0,169	0,05	0,30	0,01	99,85

Pode-se notar claramente a grande diferença nos tempos gastos nas rotinas de Teste ótimo e cálculo da direção dual em relação a estrutura de dados Indexada (Tabela 6.19). Esta diferença enfatiza mais uma vez o bom desempenho da estrutura CSC/CSR frente a Indexada para a realização das operações no cálculo da direção dual. A rotina de Teste ótimo também foi beneficiada com esta estrutura de dados e esta forma de atualização da base, e por fim, praticamente todo o tempo foi despendido para realizar o processo de atualização.

A seguir, os resultados com a estrutura CSC para a FPI são apresentados na Tabela 6.22.

Tabela 6.22: Tempo por rotinas - DSC FPI - Estrutura CSC - Instâncias Grandes

Instância	IT	Tempo Sol.(s)	Tempo Iter.(s)	Teste ótimo(%)	Dir. Dual(%)	Tam. Passo(%)	Atual.(%)
inst21_prod_2550x5150	3038	0,440	0,000	20,68	6,36	11,36	61,36
inst22_prod_3150x6190	3800	0,642	0,000	16,36	5,45	10,28	67,76
inst23_prod_3600x7170	4369	0,897	0,000	18,51	4,91	9,37	66,78
inst24_prod_3720x7380	4474	0,917	0,000	19,30	6,00	8,18	66,30
inst25_prod_3900x7700	4779	1,040	0,000	19,42	5,58	9,81	65,00
inst26_prod_4100x8180	5046	1,221	0,000	19,08	6,47	9,58	64,21
inst27_prod_4600x9190	5608	1,525	0,000	20,59	5,84	8,98	64,39
inst28_prod_5100x10200	6161	1,853	0,000	19,54	5,72	7,99	66,65
Média:	4659,375	1,067	0,000	19,20	5,80	9,44	65,307

Os resultados agora são bem diferentes para esta forma de atualização. Primeiramente como já apresentado, o tempo de solução é muito baixo, assim sendo o tempo por iteração é inferior a 10^{-3}seg. Em seguida, as demais rotinas tiveram um considerável aumento de suas porcentagens médias em relação a forma Revisada com a mesma estrutura de dados. O aumento na rotina de Teste ótimo também ocorreu em comparação com a estrutura de dados Indexada, como demonstrado no gráfico da Figura 6.7. Ocorreu também o aumento nas rotinas do cálculo da Direção dual e Tamanho do passo e mais uma vez o maior esforço está associado ao procedimento de atualização.

Na Figura 6.8 encontram-se envidenciadas as médias das porcentagens mostradas nas Tabelas 6.21 e 6.22.

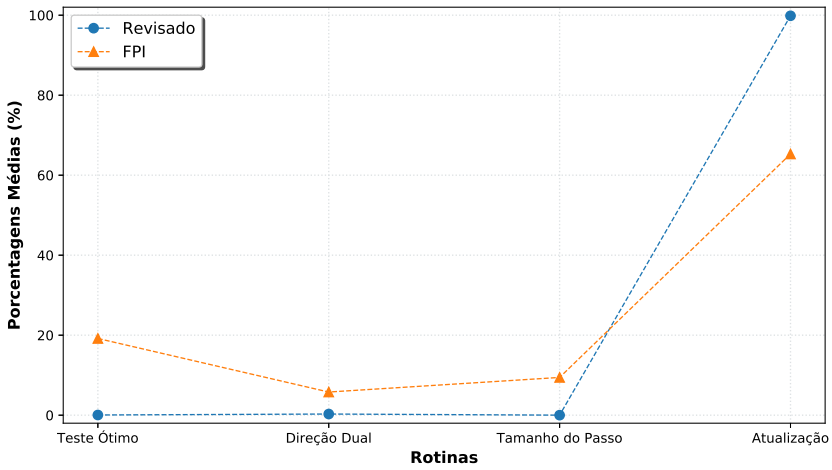

Figura 6.8: Porcentagem média das rotinas - DSC Revisado e FPI - Estrutura CSC

Com os resultados apresentados sobre o esforço computacional do DSC em relação às suas principais rotinas, fica evidenciado que o processo que exige maior tempo é o de atualização. Isto sugere que esta rotina, mesmo com estruturas esparsas continua sendo

o ponto chave de estudos futuros para possíveis otimizações no código. Também, e não menos importante, é necessário verificar com mais detalhes a rotina de Teste ótimo na abordagem FPI e tentar otimizá-la.

Capítulo 7

Conclusões e Trabalhos Futuros

A resolução de problemas de programação linear têm sido objeto de pesquisas e estudos ao longo dos anos. A contribuição das técnicas da álgebra linear e a evolução do poder de computação tornaram possíveis a abordagem de problemas cada vez maiores viabilizando assim, encontrar soluções que décadas antes eram desafios ainda não vencidos, como destacado por Bixby [4]. Ainda hoje, a implementação de um método tipo simplex estável, sobre o ponto de vista numérico, e robusta de tal modo a possibilitar a solução de diversas classes de problemas reais constitui um desafio.

As técnicas publicadas na literatura são essenciais para implementações eficientes destes métodos. Em conjunto com estruturas de dados adequadas, também representam um papel de extrema importância nesta tarefa, pois possibilitam um correto tratamento dos dados a serem processados.

Neste trabalho, o método Dual Simplex para problemas na forma geral canalizada, tratado aqui como DSC, foi implementado priorizando o processo de entrada de dados, utilizando três estruturas de dados para o armazenamento da matriz de restrições técnicas e matriz básica: Indexada, Listas Encadeadas e conjunto de vetores compactos CSC/CSR. Também foram implementadas duas formas de atualização da base: Revisada e FPI. Nos testes computacionais realizados pôde-se perceber que a estrutura de dados CSC/CSR utilizando a forma de atualização FPI obteve o melhor desempenho em todas as 3 classes de problemas abordados.

A estrutura de dados com Listas Encadeadas obteve um desempenho ruim quando utilizada em conjunto com a forma de atualização Revisada. Em contrapartida, quando implementada juntamente com a FPI, esta estrutura se mostrou bastante eficiente, chegando às vezes bem próximo do desempenho da estrutura CSC/CSR para algumas instâncias,

como pode ser visto nas Tabelas 6.7, 6.9 e 6.11.

Em relação ao número de iterações, o método DSC obteve resultados satisfatórios dado que, como já ressaltado, nenhuma técnica alternativa da escolha da variável que deixa a base foi utilizada. Para muitos dos problemas, o número de iterações chegou a ser inferior ao número de linhas do problema, como visto na Tabela 6.5, em outros casos ligeiramente superior, como sumarizado na Tabela 6.8 e, praticamente igual ao número de linhas, como exposto na Tabela 6.15. Estes resultados evidenciam mais uma vez o folclore do método Simplex no qual diz que o número de iterações é uma função polinomial de grau baixo em relação ao número de linhas do problema, Sousa(2005) [40].

Na solução de problemas reais do repositório da NetLib, o DSC também obteve um desempenho satisfatório, dado que para alguns casos encontrou a solução em um tempo bem próximo em relação a implementação do método Dual Simplex contida no *Solver* CLPEX 12.8.

Em relação aos objetivos iniciais propostos, todos foram alcançados. Com relação a trabalhos futuros, seria interessante realizar no *software* desenvolvido:

- Implementação da regra de Dantzig Normalizada (*steepest-edge rule*) para verificação de sua influência em relação ao número de iterações.
- Variações no teste da razão também poderia ser implementado, tal como teste da razão de Harris [21].
- Implementação da forma de atualização da base utilizando a abordagem de Suhl e Suhl [42] por meio da fatoração LU. Esta forma de atualização é conhecida como o estado da arte em implementações atuais e poderá, possivelmente, melhorar o tempo relacionado a este procedimento no DSC, visto que este constitui o maior esforço do método como demonstrado na seção 6.2.4 do Capítulo 6.

Referências

[1] Arenales, M. N., Armentano, V., Morabito, R., Yanasse, H. *Pesquisa Operacional*. Elsevier: ABREPO, 2011.

[2] Bartels, R. H., Golub, G. H. The simplex method of linear programming using LU decomposition. *Communications of the Association for Computing Machinery 12*, 5 (May 1969), 266–268.

[3] Bazaraa, M. S., Jarvis, J. J., Sherali, H. D. *Linear Programming and Network Flows*, 4 ed., vol. 3. John Wiley and Sons, United States, 2010.

[4] Bixby, R. Solving Real-World Linear Programs: A decade or more of progress. *Operations Research* (2002), 3–15.

[5] Bixby, R. Solving Linear and Integer Programs. Presentation on the Rice University, September 2003.

[6] Bixby, R. The Brief History of Linear and Mixed-Integer Programming Computation. *Documenta Mathematica* (2012), 107–121.

[7] Bixby, R. E. Implementing the simplex method: The initial basis. *INFORMS Journal on Computing 4* (1992), 267–284.

[8] C.A.C, K. Algebraic Languages for Mathematical Programming. *European Journal of Operational Research 67* (1993), 25–51.

[9] da Silva, C. T. L. Problemas de Otimização Linear Canalizados e Esparsos. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC - USP, São Carlos - SP, Novembro 2002.

[10] Dantzig, G. B. Maximization of a Linear Function of Variables Subject to Linear Inequalities. *Scientific Computation of Optimal Programs.* (1947).

[11] Dantzig, G. B. Reminiscences about the origins of Linear Programming. *Systems Optimizations Laboratory - Tecnical Report* (April 1981).

[12] Dantzig, G. B., Orchard-Hays. Product Form of the Inverse in the Simplex Method. *Mathematical Tables and Other Aids to Computation*, 46 (April 1954), 64–67.

[13] Dongarra, J., Sullivan, F. Guest Editor's Introduction: The top 10 Algorithms. *IEEE Computer Society Digital Library 2* (2000), 22–23.

[14] Forrest, J. J. H., Goldfarb, D. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming 57* (August 1992), 341–374.

[15] Forrest, J. J. H., Tomlim, J. A. Updated triangular factors of the basis to maintain sparsity of the product form simplex method. *Mathematical Programming 2* (September 1972), 263–278.

[16] Fourer, R. Notes on the Dual Simplex Method, March 1994. Draft Report.

[17] Gassmann, H., Ma, J., Martin, K. Instance Formats. *COINOR - Publications Manuals* (August 2009).

[18] Gill, P. E., Murray, W., Saunders, M. A., Wright, M. H. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming 45* (1989), 437–454.

[19] Grotschel, M. Solving Linear Programs. Lecture Presentation on Technical University of Berlin - Institute of Mathematics, February 2013.

[20] Hall, J. The pratical revised simplex method. Presentation on University of Edinburgh - School of Mathematics, January 2007.

[21] Harris, P. M. J. Pivot selection methods of the Devex LP code. *Mathematical Programming 5* (December 1973), 1–28.

[22] IBM. MPS file format: industry standard. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.cplex.help/CPLEX/FileFormats/topics/MPS_synopsis.html`. Acessado em 25 de Março de 2018.

[23] IBM. Passing Your Model Using Mathematical Programming System (MPS) format. `https://www.cenapad.unicamp.br/parque/manuais/OSL/oslweb/features/featur11.htm`. Acessado em 21 de Março de 2018.

[24] Júnior, P. A. M. *Theorical and computacional issues for improving the performance of linear optimization methods*. PhD thesis, Instituto de Ciências Matemáticas e de Computação - ICMC - USP, São Carlos - SP, Abril 2013.

[25] Junior, P. A. M. Técnicas computacionais para implementação eficiente e estável de métodos tipo simplex. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC - USP, São Carlos - SP, Janeiro 2009.

[26] Karmarkar, N. A new polynomial-time algorithm for linear programming. *Symposium on Theory of Computing, Washington D. C. - Combinatorica 4* (1984), 373–395.

[27] Khachiyan, L. A polinomial algorithm for linear programming. *USSR Computational Mathematics and Mathematical Physics 20* (1980), 53–72.

[28] Klee, V., Minty, G. J. How good is the simplex algorithm? *In Shisha, III Simposyum in Inequalities - UCLA* (1972), 159–175.

[29] Koberstein, A. *The Dual Simplex Method, Techniques for a fast and stable implementation*. PhD thesis, Faculty of Economics of University of Paderborn, Paderborn - Germany, November 2005.

[30] Koberstein, A., Suhl, U. H. Progress in the dual simplex method for large scale LP problems: practical dual phase 1 algorithms. *Comput Optim Appl* (2007), 37–49.

[31] Koch, T. The final Netlib-LP results. *Operations Research Letters 32* (2004), 138–142.

[32] Lemke, C. E. The Dual Method of Solving the Linear Programming Problem. *Naval Research an Logistics Quarterly.* (1954), 36–47.

[33] LPSolve. LPSolve - fixed MPS file format. `http://lpsolve.sourceforge.net/5.0/mps-format.htm`. Acessado em 25 de Março de 2018.

[34] Maros, I. *Computational Techniques of the Simplex Method*. Springer US - Kluwer Academic Publishers, 2003.

[35] Pan, P. Q. A generalization of the Revised Simplex Algorithm for Linear Programming.

[36] Rangel, S. *Introdução a Construção de Modelos de Otimização Linear e Inteira*, vol. 18. Notas em Matemática Aplicada - SBMAC, 2012.

[37] Rostagno, H., Albino, L., Donzele, J., Gomes, P., DE Oliveira, R., Lopes, D., Ferreira, A., DE Toledo Barreto, S., Euclides, R. *Tabelas Brasileiras Para Aves e Suínos: Composição de Alimentos e Exigências Nutricionais*, 3 ed. Editora UFV, 2011.

[38] Sakomura, N. K., Rostagno, H. S. *Métodos de pesquisa em nutrição de monogástricos*. Funep, 2010.

[39] Sousa, R. S. Estudos em Otimização Linear. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - ICMC USP, São Carlos - Brasil, Junho 2000.

[40] Sousa, R. S. *Métodos tipo Dual Simplex para problemas de otimização linear canalizados*. PhD thesis, Instituto de Ciências Matemáticas e de Computação - ICMC USP, São Carlos - Brasil, Outubro 2005.

[41] Stigler, G. J. The Cost of Subsistence. *Journal of Farm Economics - Oxford University Press 27* (1945), 303–314.

[42] Suhl, L. M., Suhl, U. H. A fast update LU for linear programming. *Annals of Operation Research 43*, 1 (December 1973), 33–47.

[43] Tár, P., Maros, I. Product Form of the Inverse Revisited. *The ACM Computing Classification System* (1998).

APÊNDICE A – Formato de Arquivo MPS

O formato de arquivo *Mathematical Programming System* - MPS foi introduzido pelo *mathematical programming package* da IBM nos anos 1950. Este formato reflete o estado da computação naquela época, visto que ele é baseado na premissa dos cartões perfurados, onde estes possuíam um formato fixo de 80 colunas, em que era possível armazenar um caractere em cada posição dentro de uma determinada linha, Maros(2003) [34].

Os arquivos neste padrão estão organizados de forma oposta aos formatos em que os dados estão inseridos como equações algébricas, como por exemplo, o formato LP discutido no Apêndice B, desta forma oferece pouca flexibilidade e legibilidade. Uma vez que problemas importantes no decorrer do tempo foram armazenados neste formato, ele se tornou de certa forma um padrão e a maioria dos *solvers* comerciais atuais possuem compatibilidade com ele.

O formato de arquivos descrito aqui, diz respeito ao padrão fixo, onde os nomes das linhas/colunas estão limitados a 8 caracteres. Cada linha se encontra divida exatamente em 6 campos fixos como mostrado na Tabela A.1. Comentários são representados por um símbolo de $*$ (asterisco). É importante ressaltar que neste formato, por padrão, somente elementos não-nulos são armazenados.

Tabela A.1: Organização dos campos de registros no formato MPS fixo.

Campos	**1**	**2**	**3**	**4**	**5**	**6**
Colunas	2-3	5-12	15-22	25-36	40-47	60-61
Significado	Identificador	Nome	Nome	Valor	Nome	Valor

Nem todos os Campos da Tabela A.1 são utilizados na representação de um arquivo em particular. Todos os nomes das linhas e colunas sempre devem aparecer nos campos 2, 3 e 5, e não podem começar com $ e devem estar alinhadas à esquerda. Os valores numéricos devem aparecer nos campos 4 e 6 e podem ser representados usando pontos decimais e

notação exponencial, não devem exceder 12 caracteres e não precisam necessariamente estar alinhados à esquerda.

A parte linear do padrão é organizada em seções, cada uma destas caracterizada por um identificador de registro iniciando no campo 1. Os identificadores são: `NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS` e `ENDATA`.

- `NAME`: Esta seção consiste em determinar o nome do problema a ser representado. A palavra `NAME` deve começar na coluna 1 e terminar na coluna 4. O nome do problema deve iniciar na coluna 15 e em alguns casos, pode-se estender até o final da linha. No campo campo 4 poderão aparecer os tipos `BINARY` ou `FREE` ou estar em branco. Esta seção mantém o primeiro registro da definição de um problema. A Tabela A.2 mostra sua estrutura.

Tabela A.2: Estrutura da seção `NAME` do formato MPS.

Campo-1	**Campo-2**	**Campo-3**	**Campo-4**
`NAME`	Espaço em branco	Nome do Modelo	`BINARY, FREE` ou espaço em branco

- `ROWS`: Os registros desta seção especificam cada nome que deve ser atribuído a cada linha do problema e o código que representa o tipo dos limitantes da restrição representada. O nome da função objetivo também é especificado nesta seção. Nenhum outro nome que não apareça nesta seção poderá constar na seção `COLUMNS`, caso contrário, implicará em um erro de definição. A Tabela A.3 mostra a estrutura desta seção e os tipos das restrições são mostrados na Tabela A.4.

Tabela A.3: Estrutura da seção `ROWS` do formato MPS.

Colunas de 1 a 4
`ROWS`

Tabela A.4: Tipos das restrições na seção `ROWS` do formato MPS.

Tipo da linha	**Código**	**Significado**
$\leq$	`L`	menor ou igual a
$\geq$	`G`	maior ou igual a
$=$	`E`	igual a
FO	`N`	função objetivo
Livre	`N`	restrição não vinculada

- `COLUMNS`: Nesta seção, o identificador `COLUMNS` deve aparecer nas colunas de 1 a 7. Basicamente, nesta parte os nomes das variáveis e seus respectivos valores são especificados fazendo referência as linhas da matriz de restrições e a função objetivo. Os elementos de uma determinada coluna podem aparecer em qualquer ordem, não precisam estar dispostos seguindo a sequência das linhas especificadas na seção `ROWS`, mas estes devem estar organizados em um mesmo grupo no arquivo. Os registros de dados possuem a estrutura mostrada na Tabela A.5.

Tabela A.5: Estrutura da seção `COLUMNS` do formato MPS.

Campo-1	**Campo-2**	**Campo-3**	**Campo-4**	**Campo-5**	**Campo-6**
Espaço em branco	Nome coluna	Nome 1ª linha	Valor	Nome 2ª linha	Valor

  - Campo-1: Espaço em branco
  - Campo-2: Nome da coluna (máximo de 8 caracteres)
  - Campo-3: Nome de uma das linhas que aparecem na seção `ROWS` e que possuem a coluna associada.
  - Campo-4: Valor do coeficiente representado pela coluna.
  - Campo-5: Se a mesma coluna estiver associada a outra linha, então este campo também terá o nome da próxima linha.
  - Campo-6: Coeficiente da próxima linha onde a coluna aparece.

  Os campos 5 e 6 são opcionais, pois poderá haver a situação em que uma determinada coluna não pertença a mais de uma linha, ou uma determinada coluna já estiver sido referenciada previamente em outras $n-1$ linhas na qual esta aparece e estiver restando apenas a última referência.

  Nesta seção, marcadores especiais podem aparecer para definir um conjunto de variáveis inteiras, estes marcadores devem possuir indicador de início e fim. Os marcadores devem aparecer no campo-3 e precisam ser indicados usando o seguinte padrão `'MARKER'INTORG` e `'MARKER'INTEND`, as aspas simples são obrigatórias.

- *Right Hand Side* - `RHS`: Este identificador de seção deve aparecer nas primeiras 3 colunas. Esta seção tem por objetivo definir os valores dos coeficientes do lado direito das restrições. Mais de um vetor `RHS` pode ser definido, mas somente um deve ser tomado como principal, geralmente, o primeiro a ser definido é tido como padrão. A estrutura da seção `RHS` segue o mesmo padrão mostrado na seção `COLUMNS`, com o fato de que no campo 2 encontra-se o nome do vetor.

- `RANGES`: Esta é uma seção opcional. O intervalo de uma restrição canalizada pode ser definido como:

$$L_i \leq \sum_{j=1}^{n} a_j^i x_j \leq U_i$$

Onde L_i e U_i são finitos. O intervalo da restrição $R_i = U_i - L_i$. Um dos valores (L_i ou U_i) é definido nesta seção e é denotado por b_i. Juntamente com o tipo da linha especificada na seção `ROWS`, os limitantes L_i e U_i são interpretados como mostrado na Tabela A.6.

Tabela A.6: Definição dos limitantes das restrições na seção `RANGES` do formato MPS.

Tipo da linha	**Sinal do R_i**	**Limitante inferior L_i**	**Limitante superior U_i**
L ($\leq$)	+ ou −	$b_i - \lvert R_i \rvert$	b_i
G ($\geq$)	+ ou −	b_i	$b_i + \lvert R_i \rvert$
E (=)	+	b_i	$b_i + \lvert R_i \rvert$
E (=)	−	$b_i - \lvert R_i \rvert$	b_i

Os registros desta seção são definidos da mesma maneira como apresentados na seção `RHS`.

- `BOUNDS`: Esta também é uma seção opcional. Caso esteja presente, nela estarão definidos os limitantes das variáveis/colunas do problema. Sua estrutura está organizada de forma que no campo 2 deverá estar definido o nome do vetor, no campo 3 o nome da variável/coluna e no campo 4 o valor deste limitante. Os campos 5 e 6 não são utilizados. O tipo do limitante deve possuir dois caracteres, como mostrado na Tabela A.7.

Tabela A.7: Definição dos limitantes das variáveis na seção `BOUNDS` do formato MPS.

Tipo	**Descrição**	**Significado**
`LO`	Limitante inferior	$l_j \leq x_j$
`UP`	Limitante superior	$x_j \leq u_j$
`FX`	Limitante fixo	$l_j = u_j$
`FR`	Variável livre	$-\infty \leq x_j \leq +\infty$
`MI`	Ilimitada inferiormente	$-\infty \leq x_j$
`PL`	Ilimitada superiormente	$x_j \leq +\infty$
`BV`	Variável binária	$x_j = 0$ ou $x_j = 1$

Se alguma variável não aparecer nesta seção, ou se não houver esta seção no arquivo, os limites padrão serão do tipo $0 \leq x_j \leq +\infty$.

- `ENDATA`: Este registro indica o fim da especificação do arquivo MPS.

  É importante destacar que no formato de arquivo MPS não é especificado a direção de otimização, ou seja, não é especificado se o problema deverá ser maximizado (MAX) ou minimizado (MIN).

  Para exemplificar o formato, o problema algébrico (A.1) que se encontra em Maros [34, pg. 93] foi elaborado no formato MPS e exibido na Figura A.1 destacando suas colunas e seus registros em suas devidas posições. Neste exemplo é assumido que o nome das linhas são `Res-1`, `Res-2` e `Balance` e os nomes das variáveis são `Vol--1`, `Vol--2`, `Vol--3` e `Vol--4`. A função objetivo é chamada de `OBJ`.

$$
\begin{array}{llllllll}
\max & & 4.5x1 & +2.5x2 & +4x3 & +4x4 & & \\
\text{s.t.} & & x1 & & +x3 & +1.5x4 & \leq & 40 \\
20 & \leq & & 1.5x2 & +0.5x3 & +0.5x4 & \leq & 30 \\
 & & 2.5x1 & +x2 & +3x3 & +2x4 & = & 95 \\
\multicolumn{8}{c}{x1, x2 \geq 0, -10 \leq x3 \leq 20, 0 \leq x4 \leq 25}
\end{array}
\qquad (A.1)
$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	10	21	22	23	24	25	26	27	28	29	30	31	32	33	32	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
N	A	M	E											T	e	s	t	P	r	o	b																																							
R	O	W	S											*	O	b	j	e	c	t		f	u	n	c	t	i	o	n																															
	N			O	B	J																																																					
	L			R	e	s	-	1																																																			
	L			R	e	s	-	2																																																			
	E			B	a	l	a	n	c	e																																																	
C	O	L	U	M	N	S																																																					
				V	o	l	-	-	1					O	B	J																4	.	5				R	e	s	-	1																1	.	0
				V	o	l	-	-	1					B	a	l	a	n	c	e												2	.	5																										
				V	o	l	-	-	2					O	B	J																2	.	5				R	e	s	-	2																1	.	5
				V	o	l	-	-	2					B	a	l	a	n	c	e												2	.	0																										
				V	o	l	-	-	3					O	B	J																4	.	0				R	e	s	-	1																1	.	0
				V	o	l	-	-	3					R	e	s	-	2														0	.	5				B	a	l	a	n	c	e														3	.	0
				V	o	l	-	-	4					O	B	J																4	.	0				R	e	s	-	1																1	.	5
				V	o	l	-	-	4					R	e	s	-	2														0	.	5				B	a	l	a	n	c	e														2	.	0
R	H	S																																																										
				R	H	S	-	1						R	e	s	-	1													4	0	.	0				R	e	s	-	2															3	0	.	0
				R	H	S	-	1						B	a	l	a	n	c	e											9	5	.	0																										
R	A	N	G	E	S																																																							
				R	A	N	G	E	-	1				R	e	s	-	2														1	0	.	0																									
B	O	U	N	D	S																																																							
	L	O		B	O	U	N	D	-	1				V	o	l	-	-	3												-	1	0	.	0																									
	U	P		B	O	U	N	D	-	1				V	o	l	-	-	3													2	0	.	0																									
	U	P		B	O	U	N	D	-	1				V	o	l	-	-	4													2	5	.	0																									
E	N	D	A	T	A																																																							

Figura A.1: Exemplo `TestProb` no formato MPS fixo por colunas.

Mais informações sobre o formato MPS podem ser encontradas em IBM MPS File [22], IBM OSL [23] e LPSolve [33]

APÊNDICE B – Formato de Arquivo LP

Problemas de PL descritos no formato LP são representados de maneira algébrica orientado por linhas, quase da mesma maneira como são descritos nos livros, relatórios e artigos sobre PL. Este formato tem a vantagem de ser mais compreensível sobre o ponto de vista de leitura e *debuggind* [17]. Muitos *solvers* possuem suportam algum tipo similar ao formato algébrico, por exemplo, o CPLEX LP File format[1], LPSolve *lp-format*[2] e o XPress lp *format*[3]. Neste apêndice, particularmente o padrão abordado será o formato utilizado pelo CPLEX 12.8, pois foi este o padrão utilizado na implementação do código do presente trabalho.

O formato ILOG CPLEX LP consiste de uma alternativa ao formato MPS. Neste formato, um problema LP é descrito pela seguinte sequência: Função Objetivo (FO), Restrições e Limitantes das variáveis. Para que esta sequência esteja corretamente disposta, alguns marcadores de seção são utilizados, sendo estes: `Minimize/Maximize`, `Subject To`, `Bounds` e `End`.

A sintaxe padrão do formato CPLEX LP não é *case sensitive*, ou seja, maiúsculas e minúsculas não fazem diferença nas definições de seções e nomes presentes no arquivo. Comentários são representados por '\'(barra invertida ou *backslash symbol*).

- Função Objetivo: Para a definição da FO no formato LP, inicialmente é necessário descrever a direção na qual o problema será resolvido. Isto é informado fazendo o uso das palavras reservadas `Minimize` para minimização e `Maximize` para maximização. Ambas palavras podem ser abreviadas para `Min` e `Max`, respectivamente.

- Nomes de Variáveis: Variáveis podem conter no máximo 255 caracteres e não devem começar com número. Os nomes também não podem conter a letra '`e`' ou '`E`' de

[1] `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/FileFormats/topics/LP.html`

[2] `http://lpsolve.sourceforge.net/5.1/lp-format.htm`

[3] `http://lpsolve.sourceforge.net/5.5/Xpress-format.htm`

maneira isolada ou seguida de um número ou letra, pois esta forma é utilizada para entrada de números com notação exponencial, como, por exemplo $2e3$, significando 2×10^3.

- Restrições: A seção relacionada as restrições deve começar com a palavra reservada `Subject To` ou de forma abreviada `st`. Cada restrição deverá possuir no final um dos símbolos ($\leq, \geq ou =$) e em seguida conter um valor numérico correspondendo ao respectivo limitante. Para inserir uma restrição canalizada superior e inferiormente, esta deverá ser repetida no arquivo, sendo que uma ocorrência deverá fazer referência ao limitante superior e a outra ao limitante inferior. Caso algum dos limitantes (inferior ou superior) não estejam presentes, serão atribuídos os valores $-\infty$ para limitante inferior e $+\infty$ para o limitante superior automaticamente. As restrições e a FO podem ser nomeadas, isto pode ser feito adicionando-se um nome antes da declaração. O nome deve seguir as mesmas regras de nomeação de variáveis com o seguinte padrão `<nomeRestricao>:`. Caso nenhum nome seja atribuído a FO, o ambiente do CPLEX 12.8 irá renomeá-la para '`obj`' automaticamente.

- Limitantes das Variáveis: Os limitantes são especificados na seção `BOUNDS`. Diferentemente da seção `st`, para a representação de uma variável canalizada superior e inferiormente, não é necessário repetí-la, bastando apenas informá-la no seguinte padrão `<limInf>` $\leq$ `<nomeVariavel>` $\leq$ `<limSup>`. Se uma variável não possuir algum dos limitantes inferior ou superior, ou ambos, de maneira explícita, então automaticamente serão atribuídos os valores 0 para inferior e $+\infty$ o superior de automaticamente.

Para finalizar a descrição do problema no arquivo, este deve-se terminar com a palavra reservada `end`.

Com o objetivo de ilustrar o uso do formato LP, o exemplo algébrico apresentado em (A.1) é mostrado em (B.1) utilizando as regras anteriormente citadas.

$$
\begin{array}{llllllll}
\texttt{Maximize} & & & & & & & \\
Obj: & 4.5x1 & +2.5x2 & +4x3 & +4x4 & & & \\
\texttt{Subject To} & & & & & & & \\
Res1: & x1 & & +x3 & +1.5x4 & \leq & 40 & \\
Res2: & & 1.5x2 & +0.5x3 & +0.5x4 & \leq & 30 & \\
Res2: & & 1.5x2 & +0.5x3 & +0.5x4 & \geq & 20 & \\
Balance: & 2.5x1 & +2x2 & +3x3 & +2x4 & = & 95 & \\
\texttt{Bounds} & & & & & & & \\
-10 & \leq & x3 & \leq & 20 & & & \\
x4 & \leq & 25 & & & & &
\end{array}
\tag{B.1}
$$

$$
\texttt{End}
$$

APÊNDICE C – Parâmetros Modificados do Solver CPLEX 12.8

O *solver* comercial CPLEX 12.8 foi utilizado como parâmetro para comparação em relação a implementação do DSC. Para uma comparação justa, alguns parâmetros de configuração foram desabilitados. Abaixo, segue uma breve descrição destes parâmetros retirados do manual de referência de parâmetros do CPLEX V12 Release 6[1].

- CPXPARAM_Advance: Se definido como 1 ou 2, esse parâmetro especifica que o CPLEX deve usar informações para inicialização avançada ao iniciar a otimização. A Tabela C.1 mostra as possíveis configurações de valores deste parâmetro.

  Tabela C.1: Possíveis valores de configuração do CPXPARAM_Advance.

Valor	**Significado**
0	Não utiliza inicialização avançada
1	Utilizar base avançada fornecida pelo usuário (**Padrão**)
2	Utilizar um vetor fornecido pelo usuário

  O parâmetro foi configurado com o valor 0 nos testes realizados.

- CPXPARAM_Preprocessing_Aggregator: Invoca o agregador para usar substituição sempre que possível para reduzir o número de linhas e colunas antes que um problema seja resolvido. Se for definido um valor positivo, o agregador é aplicado o número especificado de vezes ou até não ser mais possível realizar reduções.

  A Tabela C.2 mostra as possíveis configurações de valores deste parâmetro.

[1] https://ibm.co/2OQnpXA

Tabela C.2: Possíveis valores de configuração do CPXPARAM_Preprocessing_Aggregator.

Valor	**Significado**
-1	Automático (1 para LP e "infinito" para MIP) (**Padrão**)
0	Não utilizada agregador
Qualquer inteiro positivo	Número de vezes a aplicar o agregador

O parâmetro foi configurado com o valor 0 nos testes realizados.

- CPXPARAM_Simplex_DGradient: Este parâmetro é utilizado para decidir o tipo de regra de *pricing* a ser adotada em um problema utilizando o método Dual Simplex.

  A Tabela C.3 mostra as possíveis configurações de valores deste parâmetro.

Tabela C.3: Possíveis valores de configuração do CPXPARAM_Simplex_DGradient.

Valor	**Significado**
0	Automático. Permite o CPLEX escolher. (**Padrão**)
1	Dual *pricing* padrão
2	*Steepest-edge pricing*
3	*Steepest-edge pricing* com *slack spacing*
4	*Steepest-edge pricing* com normas iniciais
5	*Devex pricing*

O parâmetro foi configurado com o valor 1 nos testes realizados.

- CPXPARAM_Preprocessing_Presolve: Utilizado para decidir se o CPLEX irá aplicar o processo de *presolve* durante a solução de algum problema.

  A Tabela C.4 mostra as possíveis configurações de valores deste parâmetro.

Tabela C.4: Possíveis valores de configuração do CPXPARAM_Preprocessing_Presolve.

Valor	**Significado**
0	Não aplica *presolve*
1	Aplica *presolve*. (**Padrão**)

O parâmetro foi configurado com o valor 0 nos testes realizados.

- CPXPARAM_Preprocessing_NumPass: Quando este parâmetro é definido com um valor positivo, o *presolve* é aplicado de acordo com o número de vezes especificado, ou até que nenhuma outra redução seja possível.

  A Tabela C.5 mostra as possíveis configurações de valores deste parâmetro.

Tabela C.5: Possíveis valores de configuração do CPXPARAM_Preprocessing_NumPass.

Valor	**Significado**
-1	Automático. Permite CPLEX continuar com *presolve* enquanto for permitido.(**Padrão**)
0	Não usa *presolve*, porém outras reduções podem ocorrer
Qualquer inteiro positivo	Aplica *presolve* com base no número de vezes informado

O parâmetro foi configurado com o valor 0 nos testes realizados.

- CPXPARAM_Preprocessing_Reduce: Especifica se haverá reduções primal, dual, ou ambas durante a realização do pré-processamento.

  A Tabela C.6 mostra as possíveis configurações de valores deste parâmetro.

Tabela C.6: Possíveis valores de configuração do CPXPARAM_Preprocessing_Reduce.

Valor	**Significado**
0	Sem reduções primais ou duais
1	Somente reduções primais
2	Somente reduções duais
3	Reduções primais e duais. (**Padrão**)

O parâmetro foi configurado com o valor 0 nos testes realizados.

- CPXPARAM_Preprocessing_Linear: Utilizado para decidir se irão ocorrer reduções lineares ou totais durante o *presolve*. Se apenas reduções lineares são realizadas, então cada variável no modelo original pode ser expressa como uma combinação linear de variáveis no modelo pós *presolve*.

  A Tabela C.7 mostra as possíveis configurações de valores deste parâmetro.

Tabela C.7: Possíveis valores de configuração do CPXPARAM_Preprocessing_Linear.

Valor	**Significado**
0	Executa somente reduções lineares
1	Executa reduções completas. (**Padrão**)

O parâmetro foi configurado com o valor 0 nos testes realizados.