Universidade Federal Fluminense

ALLANDERSON RODRIGUES TEIXEIRA

Algoritmos do tipo *Local Branch* e Genético de Chaves Aleatórias Viciadas aplicados ao Problema de Distribuição de Capacidades, Réplicas e Requisições

VOLTA REDONDA

2022

ALLANDERSON RODRIGUES TEIXEIRA

Algoritmos do tipo *Local Branch* e Genético de Chaves Aleatórias Viciadas aplicados ao Problema de Distribuição de Capacidades, Réplicas e Requisições

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: Pesquisa Operacional.

Orientador:

Tiago Araujo Neves

Coorientador:

Emerson de Souza Freire

Universidade Federal Fluminense

VOLTA REDONDA

2022

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

T266a Teixeira, Allanderson Rodrigues
Algoritmos do tipo Local Branch e Genético de Chaves Aleatórias Viciadas aplicados ao Problema de Distribuição de Capacidades, Réplicas e Requisições / Allanderson Rodrigues Teixeira. - 2022.
106 f.: il.

Orientador: Tiago Araujo Neves.
Coorientador: Emerson de Souza Freire.
Dissertação (mestrado)-Universidade Federal Fluminense, Escola de Engenharia Industrial e Metalúrgica de Volta Redonda, Volta Redonda, 2022.

1. Redes de Dados Distribuídos. 2. Problema de Distribuição de Capacidades, Réplicas e Requisições. 3. Algoritmo Local Branch. 4. Algoritmo Genético de Chaves Aleatórias Viciadas. 5. Produção intelectual. I. Neves, Tiago Araujo, orientador. II. Freire, Emerson de Souza, coorientador. III. Universidade Federal Fluminense. Escola de Engenharia Industrial e Metalúrgica de Volta Redonda. IV. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Algoritmos do tipo *Local Branch* e Genético de Chaves Aleatórias Viciadas aplicados ao Problema de Distribuição de Capacidades, Réplicas e Requisições de Distribuição de Capacidades, Réplicas e Requisições

Allanderson Rodrigues Teixeira

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: Pesquisa Operacional.

Aprovada por:

Prof. Tiago Araujo Neves, D.Sc. / MCCT-UFF (Orientador)

Emerson Souza Freire

Prof. Emerson de Souza Freire, D.Sc. / MCCT-UFF (Coorientador)

Prof. Luiz Satoru Ochi, D.Sc. / IC-UFF

Prof. Adriana de Cesário de Faria Alvim, D.Sc. / UNIRIO

Prof. Cecilia Toledo Hernandez, D.Sc. / MCCT-UFF

Volta Redonda, 28 de outubro de 2022.

A Deus, a meus filhos e à minha família.

Agradecimentos

A Deus, criador de todas as coisas, o grande arquiteto do universo, o qual tem cuidado, orientado e sustentado minha vida, meus filhos e minha família, norteando minhas escolhas e permitindo traçar minha carreira profissional, concluindo mais um desafio, que é me tornar mestre em Modelagem Computacional nesta renomada instituição de ensino.

Aos meus filhos, Alisson e Esther, razão dos meus esforços e da minha vida, elementos que me dão suporte para prosseguir, para acreditar que a vida vale a pena e que necessitamos contribuir e construir uma sociedade mais justa e melhor.

Aos meus pais, Joaquim e Eni (*in memorian*), por toda educação, dedicação, empenho e sacrifícios que desprenderam a minha criação e a dos meus irmãos, abdicaram-se do conforto e do descanso para que pudessem oferecer o melhor, criando condições para eu chegar até aqui.

À minha amada irmã Amandha (*in memorian*), vítima da COVID-19, obrigado por tudo e que Deus a tenha em uma boa jornada, o meu até breve.

Ao meu orientador, Prof. D.Sc. Tiago, obrigado pela confiar a mim a continuidade do desenvolvimento do trabalho abordado em sua tese; suas orientações facilitaram, nortearam e permitiram a realização da minha dissertação.

Ao Exército Brasileiro, na figura da Academia Militar das Agulhas Negras, que proporcionou condições e apoio para a realização do curso.

Aos meus professores desta renomada instituição de ensino, André Gusso, Cleyton Senior Stampa, Diomar Cesar Lobão, Emerson de Souza Freire, Gustavo Benitez Alvarez, Luis Alberto Duncan Rangel, Nirzi Goncalves de Andrade, Tiago Araújo Neves e Wellington Gomes Dantas. Registro minha eterna gratidão pelos conhecimentos compartilhados e que sobremaneira contribuíram para a minha formação como mestre.

Aos amigos e familiares pelo apoio, paciência e a compreensão nos momentos de ausência que a pesquisa e os estudos requerem.

Resumo

Com o advento e o uso massivo da Internet, a quantidades de dados e informações compartilhadas tem crescido aceleradamente, elevando o número de requisições aos serviços digitais, elevando a carga de seus servidores e dos meios de comunicação, motivando o delineamento de novas técnicas de entrega de conteúdo que exploram suas características específicas. Com isso, a distribuição de conteúdo na Internet tem migrado de uma arquitetura onde os dados são armazenados em um único servidor para uma arquitetura onde os dados são replicados em servidores distribuídos geograficamente, as chamadas Redes de Distribuição de Conteúdos ou *Content Distribution Networks* (CDN). A CDN é uma rede informatizada, sobreposta e de alto desempenho que mantém réplicas de cada conteúdo em seus servidores, tendo por finalidade a redução de congestionamentos nas redes digitais e minimização das cargas nos servidores. Neste contexto existem diversos problemas estudados na literatura, entre eles está Problema de Distribuição de Capacidades, Réplicas e Requisições (PDCRR), que trata simultaneamente o Problema de Posicionamento das Réplicas e Distribuição de Requisições (PPRDR) e o Problema de Alocação da Capacidade de Armazenamento (PACA). O método exato para resolução do modelo matemático do PDCRR não obteve êxito na busca por soluções ótimas em alguns casos, mostrando que este é um problema de difícil solução, principalmente quando o número de servidores é elevado. Neste trabalho será implementada a técnica de *local branching*, que prioriza a exploração parcial, ou completa da vizinhança de soluções viáveis na busca por soluções ótimas do problema. Além desta técnica, também será utilizado o algoritmo genético de chaves aleatórias viciadas como mecanismo de resolução. Os resultados computacionais são comparados com o método exato, mostrando que mesmo fazendo explorações parciais da vizinhança, o PDCRR é um problema de árdua solução, e que a abordagem heurística é a melhor opção para instâncias maiores do problema.

Abstract

With the advent and massive use of the Internet, the amount of data and information shared has grown rapidly, increasing the number of requests for digital services, increasing the load on servers and the media, motivating the design of new delivery techniques of content that explore their specific characteristics. As a result, content distribution on the internet has migrated from an architecture where data is stored on a single server to an architecture where data is replicated on geographically distributed servers, the so-called Content Distribution Networks (CDN) . The CDN is a computerized, overlapping, high-performance network that maintains replicas of each content on its servers, with the aim of reducing congestion in digital networks and minimizing server loads. In this context, there are several problems studied in the literature, among them is the Capacity, Replicas and Requests Distribution Problem, which simultaneously deals with the Replicas Positioning and Request Distribution Problem and the Storage Capacity Allocation Problem. The exact method for solving the PDCRR mathematical model was not successful in the search for optimal solutions in some cases., showing that this is a difficult problem to solve, especially when the number of servers is high. In this work, the local branching technique will be implemented, that prioritizes partial or complete exploration of the neighborhood of viable solutions in the search for optimal solutions to the problem. In addition to this technique, the genetic algorithm of biased random keys was also used as a resolution mechanism. The computational results are compared with the exact method, showing that even making partial explorations of the neighborhood, the PDCRR is an arduous problem, and that the heuristic approach is the best option for larger instances of the problem.

Palavras-chave

1. Redes de Distribuição de Conteúdos
2. Qualidade de Serviço
3. Programação Inteira Mista
4. Problema de Posicionamento de Réplicas e Distribuição de Requisição
5. Problema da Alocação de Capacidade de Armazenamento
6. Problema de Distribuição de Capacidades, Réplicas e Requisições
7. Formulação Matemática
8. *Local Branching*
9. Algoritmo Genético de Chaves Aleatórias Viciadas

Glossário

PLS	:	Problema de Localização de Servidores
PR	:	Ploblema de Replicação
PPR	:	Problema de Posicionamento de Réplicas
PDR	:	Problema de Distribuição de Requisições
PDCR	:	Problema de Distribuição de Capacidades e Réplicas
PPRDR	:	Problema de Posicionamento de Réplicas e Distribuição de Requisições
PACA	:	Problema da Alocação de Capacidade de Armazenamento
PFCM	:	Problema de Fluxo de Custo Mínimo
PDCRR	:	Problema de Distribuição de Capacidades, Réplicas e Requisições
CDN	:	Content Distribution Networks
URL	:	Uniform Resource Locator
QoS	:	Quality of Service
PPL	:	Problema de Programação Linear
PIM	:	Programação Linear Intera Mista
BRKGA	:	Biased Random Key Genetic Algorithm
VNS	:	Variable Neighbourood Search
RINS	:	Relaxation Induced Neighbourood Search
VND	:	Dariable Neighborhood Descent

Sumário

Capítulo 1

INTRODUÇÃO

Com o advento da Internet e o uso massivo das redes de computadores, a quantidades de dados e informações compartilhadas, por meio desta rede, tem crescido aceleradamente. Estes compartilhamentos de informações variam desde bancos de dados, redes sociais até *streaming* de áudios e vídeos. A quantidade de interações por usuário referentes ao período de 01 hora, 01 dia e 01 mês em redes sociais como *Facebook* e *Instagram*, novos *tweetts*, vídeos assistidos no *YouTube*, mensagens no *WhatsApp* seguem os quantitativos conforme apresentado na Tabela 1.1, dados que foram obtidos no *site BETFYUK* [2].

Tabela 1.1: Variação temporal da quantidade de interações por usuário em redes digitais de relacionamento e entretenimento.

Rede	Tempo		
	1 hora	1 dia	1 mês
Facebook	249998400	06 bilhões	180 bilhões
Instagram	174999600	04 bilhões	126 bilhões
Twitter	26107200	629572800	19 bilhões
WhatsApp	1000000000	30 bilhões	899 bilhões
Youtube	451461600	11 bilhões	325 bilhões

O grande número de requisições aos serviços digitais dos servidores das redes sociais, sites e aplicativos tem elevado a carga de seus servidores e dos meios de comunicação, impondo aos usuários tempos de respostas cada vez mais elevados. O problema desta lentidão nas respostas das ferramentas digitais de comunicação e informação se torna ainda mais desafiador se considerarmos os requisitos crescentes para uma variedade de novas aplicações, personalizações e, mais recentemente, a mobilidade.

Em particular, à medida que as redes móveis de alta largura de banda se tornam

disponíveis, a implantação destas redes como um meio de disseminação de informações em uma empresa aumenta em popularidade, motivando o delineamento de novas técnicas de entrega de conteúdo que exploram suas características específicas.

Com isso, a distribuição de conteúdo na internet tem migrado de uma arquitetura onde os objetos são armazenados em um único servidor para uma arquitetura onde os objetos são replicados em servidores distribuídos geograficamente, as chamadas Redes de Distribuição de Conteúdos, em Inglês *Content Distribution Networks* ou CDN.

1.1 Redes de Distribuição de Conteúdos

A Figura 1.1 ilustra uma CDN, um sistema de computadores interligados por meio da Internet e que colaboram para fornecer conteúdos digitais para os usuários. A CDN é uma rede informatizada, sobreposta e de alto desempenho que mantém réplicas de cada conteúdo em seus servidores, tendo por finalidade a redução de congestionamentos nas redes digitais e minimização das cargas nos servidores, Neves [44].

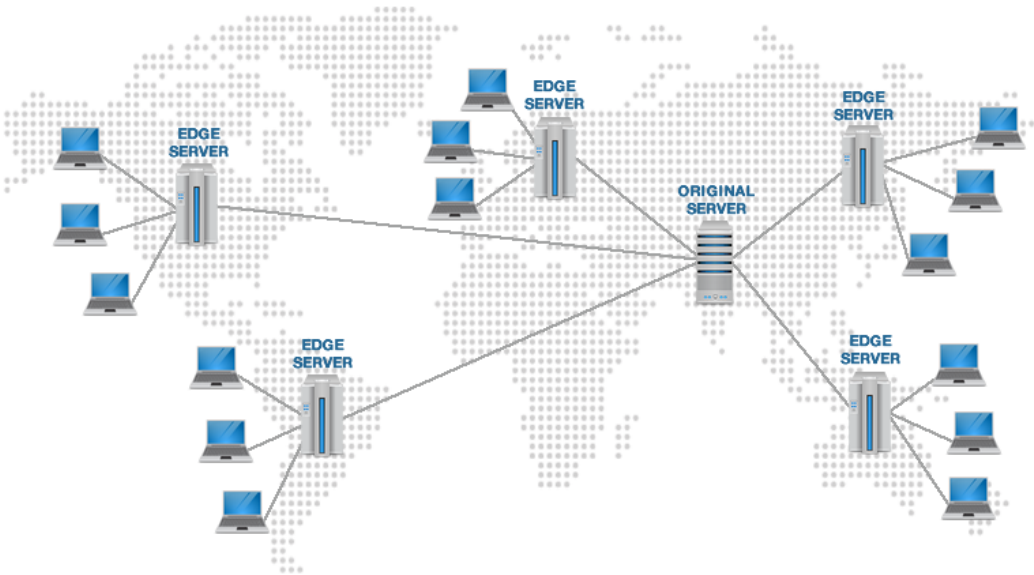

Figura 1.1: Exemplo de uma RDC em escala global, RDC para *WordPress* [6].

A CDN atua como intermediária entre os usuários e os servidores. Em geral, os objetos incorporados a uma rede digital estão localizados no mesmo servidor que o documento base, mas isso não é realmente uma necessidade. Os objetos incorporados também são descritos por meio de URLs, em inglês *Uniform Resource Locator*, o que possibilita que

eles estejam localizados em um servidor diferente do documento base, Neves [44].

Segundo o *site Windows Azure* [7], a Figura 1.2 demostra o modelo de atendimento proposto pela CDN, o usuário final utiliza o navegador (por exemplo) para requerer um determinado conteúdo, a aplicação que utiliza CDN propositalmente busca o conteúdo solicitado no nó mais próximo, se o conteúdo estiver disponível, os dados são automaticamente repassados para o usuário final. Se o conteúdo não estiver diponível na CDN, a aplicação carrega o conteúdo na fonte original, coloca uma cópia na CDN, devolve a resposta para o usuário final e a partir da próxima requisição ao conteúdo recém baixado, a resposta devolvida será o conteúdo residente na CDN.

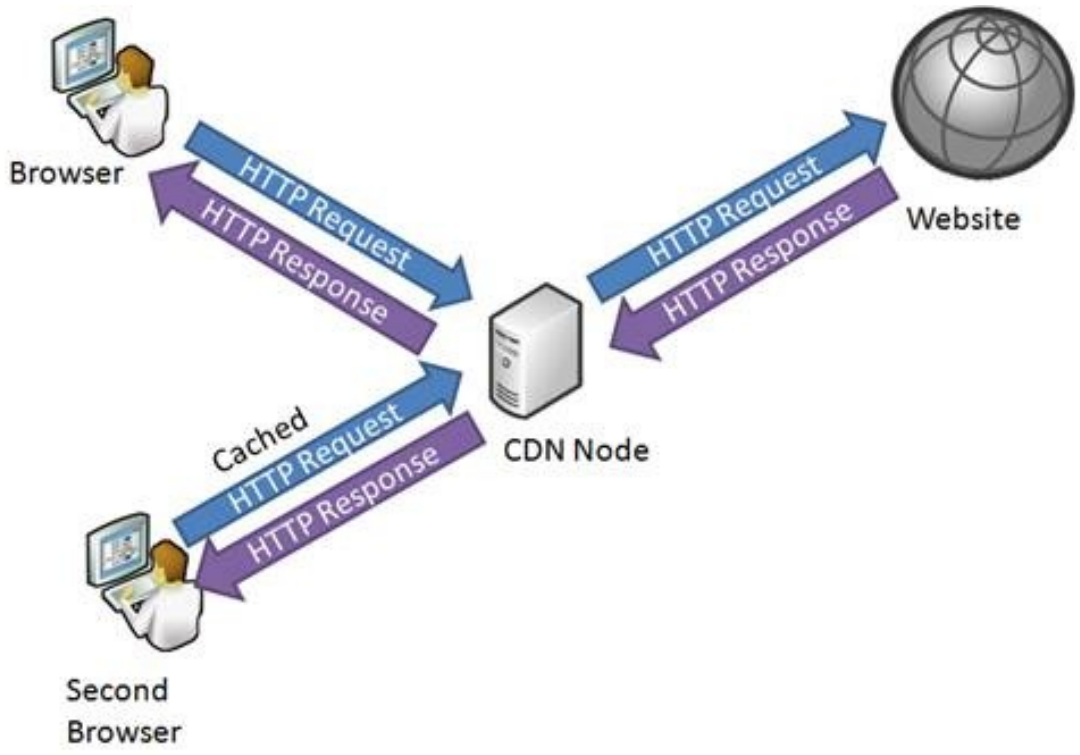

Figura 1.2: O modelo de computação implementado por RDC, *Windows Azure*[7].

Com isso, a CDN replica os objetos incorporados, também chamados de conteúdos, e posiciona estrategicamente estas réplicas em servidores próximos aos usuários, procurando com isso reduzir o atraso percebido por estes no uso das redes digitais. A redução da carga de trabalho nos servidores é um dos resultados da existência de múltiplas réplicas de um conteúdo posicionadas em diferentes servidores, levando em conta que existe a possibilidade de atendimento a uma fração do total de demandas pelos servidores, Xu et al. [58].

De acordo com Neves [44], a Figura 1.3 mostra o processamento de uma requisição em uma CDN. Normalmente, quando queremos acessar um conteúdo na Internet, digitamos

o endereço virtual da página, URL, o navegador utiliza um serviço de rede chamado DNS, *Domain Name System* ou Sistema de Nomes de Domínios, que é responsável por realizar o mapeamento da URL em um endereço IP do *site* requisitado, Filippetti [24]. Em uma CDN o funcionamento segue como ilustrado na Figura 1.3, cujos passos serão descrito de acordo com a numeração contida na mesma:

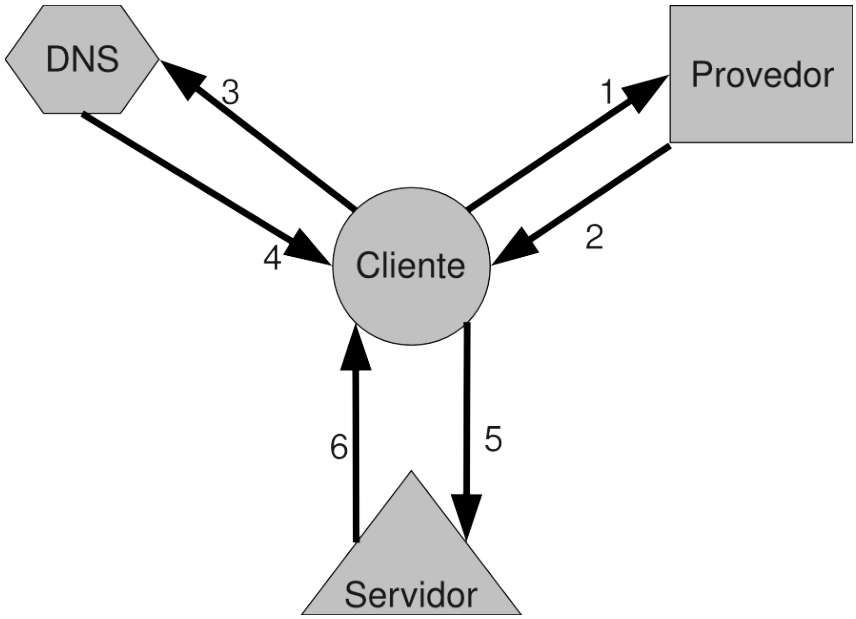

Figura 1.3: Processamento de uma requisição em uma RDC, Neves [44].

1. O cliente envia uma requisição para um determinado endereço virtual, como se o conteúdo não estivesse replicado pela CDN;
2. Como o conteúdo está replicado pela CDN, o servidor envia para o cliente as URLs modificadas de acordo com a localização do conteúdo na CDN;
3. O cliente de posse da URL recebida, consulta o servidor DNS, para receber o endereço IP do servidor que armazena o conteúdo;
4. O servidor DNS analisa a requisição e fornece o endereço IP do servidor mais próximo da origem da requisição que contenha a réplica do conteúdo solicitado pelo cliente;
5. De posse do endereço do servidor o cliente encaminha a sua requisição;
6. O servidor responde à requisição do cliente.

Conforme descrito acima, a URL de cada um dos objetos que serão distribuídos pela CDN é alterada de modo que ela não aponte para um servidor controlado pelo fornecedor do conteúdo, mas sim para um que seja controlado pela operadora da CDN. Para

determinar qual servidor utilizar, uma função de *hash*, que considera vários parâmetros, é aplicada ao conteúdo para escolher um dos servidores da operadora de CDN. O nome deste servidor é então colocado na frente da URL original fazendo com que, ao solicitar o conteúdo, os clientes o façam para um dos servidores controlados pela operadora de CDN, e não para um servidor controlado pelo fornecedor do conteúdo, Neves et al. [42] e Xu et al. [58].

Por meio das CDN, os clientes acessam transparentemente a cópia mais próxima do objeto desejado, de modo que não percebam esta distribuição física da rede. Assim, do ponto de vista do usuário, as informações são originárias de uma única fonte e isso melhora o desempenho das aplicações de tal modo que é viabilizado o balanceamento do acesso dos usuários aos conteúdos armazenados nos servidores das redes digitais, Garg et al. [27].

A CDN deve possuir garantia de entrega de conteúdo aos usuários. Essa garantia é alcançada quando os servidores são alocados em distintas regiões para evitar problemas de indisponibilidade da rede. Além de garantir a entrega de conteúdos, uma CDN ideal deve reproduzir com assertividade os conteúdos dentro da rede, assim, o desempenho de uma CDN está em função do posicionamento dos conteúdos, alocação dos servidores, estratégias de replicação e armazenamento digital, Boechat [18].

De acordo com Goldbach [29], a CDN pode ser utilizada na modelagem computacional para solucionar os problemas decorrentes do intenso compartilhamento de informações nas redes digitais. Segundo Goldbach [29], uma CDN pode posicionar uma ou mais réplicas de conteúdos em alguns servidores para minimizar o congestionamento na rede digital e reduzir atrasos na entrega dos conteúdos utilizados pelos usuários. Os atrasos aqui abordados são aqueles referentes ao atendimento de uma requisição a partir de um servidor mais distante do cliente, porém com mais banda disponível.

Entre os parâmetros que podem ser considerados na função *hash* estão a data de criação do objeto, tamanho do objeto, sua popularidade antecipada e seu formato (foto, vídeo, etc) e o conteúdo em si. Com estas informações, é possível balancear a carga, direcionar o fluxo de um determinado tipo para um conjunto específico de servidores e ainda evitar que o cliente receba conteúdo desatualizado, Neves et al. [42].

É importante mencionar que estes servidores são virtuais, ou seja, eles não são máquinas físicas. Uma mesma máquina real pode abrigar diversos servidores virtuais dependendo de sua capacidade. A virtualização dos servidores é uma característica que aumenta a tolerância a falhas da estrutura como um todo, pois, ao perceber que um servidor virtual não pode ser encontrado, a estruturada CDN pode encontrar outro servidor

virtual utilizando os parâmetros da função de *hash*, Neves [44].

No decorrer de um atendimento por CDN, os servidores que se encontram mais próximos a um usuário podem encerrar a transmissão de outros conteúdos tornando-se aptos a atender aquele usuário. Com isso, pode ocorre uma mudança no atendimento, e neste sentido, o usuário passa a ser atendido por outros servidores pelo restante do tempo da conexão. Este tipo de mudança de servidor pode ser feita durante o processo de atendimento de todas as requisições e racionaliza o uso da estrutura da CDN, pois melhora a qualidade da rede digital e reduz o tráfego na rede, Ruiz [36].

Na atualidade, segundo o *site Cloud CDN Google* [5], empresas como AKAMAI, Google, Amazon, Coudflare e Azure, entre outras, atuam a vários anos no mercado em múltiplos países. De acordo com o *site Akamai Technologies* [3], as CDNs são o pilar invisível da Internet, entregando conteúdos *on-line* para empresas de compras, serviços bancários, saúde e outras com velocidade e em escala, na qual sem as CDNs, com a capacidade de replicar e armazenar informações de servidores de origem e, em seguida, trazer o conteúdo digital para perto de onde os usuários acessam a rede, a Internet poderia ficar congestionada e haveria lentidão na entrega dos conteúdos e serviços. Ainda de acordo com *site Akamai Technologies* [3], todos que acessam a rede usam uma CDN, utilizando uma experiência mais rápida e mais confiável, usufruindo dos conteúdos e aplicações disponibilizadas por provedores de serviços de rede, onde os proprietários de conteúdos e aplicações, como *websites* de comércio eletrônico, propriedades de mídia e empresas de computação em nuvem, usam CDNs para aprimorar a experiência do cliente, reduzir as taxas de abandono, aumentar as impressões de anúncios, melhorar as taxas de conversão e fortalecer a fidelidade dos clientes. Outro aspecto citado pelo *site Akamai Technologies* [3] é a melhora na segurança na rede, ajudando a absorver e mitigar ataques de negação de serviço.

Outro exemplo atual de utilização de CDN é o *Cloud CDN Google* [5] a qual usa a rede de borda global do Google para exibir conteúdo mais relevante aos usuários, o que acelera seus *sites* e aplicativos. A *Cloud CDN* funciona com o balanceamento de carga externo para fornecer conteúdo aos usuários. Segundo o *site CDN Peering* [4], uma *Cloud CDN* funciona quando um usuário solicita conteúdo de um balanceador de carga externo, e a solicitação chega a um *Google Front End*, (GFE), que fica na borda da rede do Google, o mais próximo possível do usuário. Se o mapa de URLs do balanceador de carga direcionar o tráfego para um serviço que tenha a *Cloud CDN* configurado, o GFE usará a *Cloud CDN*.

O *site* da empresa Akamai [4] destaca que apesar dos benefícios mútuos para clientes e ISP, *Internet Service Provider*, empresas ou corporações que fornecem às pessoas acesso à Internet a um preço, e até mesmo para a detentora da tecnologia, empresa de conteúdo, que consegue engajar e monetizar melhor seus clientes, ter uma CDN não é direito, é privilégio. As CDNs, baseadas em critérios próprios, definem condições em que um servidor pode ser enviado para ser instalado dentro do *backbone* de um ISP para otimizar a entrega de conteúdo para o usuário final. Isso é vantajoso para a CDN que melhora a qualidade na entrega do conteúdo e também para ISP pois isso melhora a experiência para os usuários finais e reduz o fluxo de dados, gerando economia. Ainda de acordo *site* da empresa Akamai [4], o tráfego atual de referência para solicitação de servidores de algumas CDN está na faixa dos 2 a 5 Gbps, porém existem condições que isso pode ser flexibilizado à depender da região e da disponibilidade de outros trânsitos IP presentes serem capazes de distribuir aquele conteúdo. Via de regra em uma grande cidade ou região populosa que já possui servidores de cache ou uma quantidade razoável de trânsitos IP é bastante provável que o valor de referência não seja flexibilizado pelas CDNs e cita as seguintes empresas que utilizam CDNs no território nacional: Akamai, Apple, Facebook, Google, Edgecast Verizon Digital Media, Netflix, Azion, SourceForge.Net, Microsoft e CloudFlare.

1.2 Problema de Distribuição de Capacidades, Réplicas e Requisições

Os principais problemas de otimização nas CDN consistem no Problema de Localização de Servidores (PLS), Problema de Replicação (PR), Problema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR), Aioffi et al. [11] e Zhou et al. [59].

O Problema de Localização de Servidores pertence à classe NP-Difícil, *NP-Hard* Bektas et al. [14], e consiste em encontrar os melhores locais dentro da rede real para posicionar os servidores que irão compor a CDN e assim minimizar o tráfego de informações.

O Problema de Replicaçao pertence à classe P, Zhou et al. [59], e consiste em decidir quais conteúdos serão replicados. A replicação de dados é um dos componentes de sistemas redundantes, na qual mantém uma ou mais cópia de dados ou sistema em outro local físico ou lógico, no caso de servidores virtualizados. Além disso, sistemas distribuídos e de alta tolerância a falhas buscam a replicação de suas plataformas para atingir melhores desempenhos.

O Problema de Distribuição de Requisições, na qual também pertence a classe P segundo Aioffi et al. [11] e Zhou et a. [59], consiste em, dado um conjunto de servidores com réplicas posicionadas, encontrar a melhor distribuição das requisições para o conjunto de servidores de modo a reduzir os custos de transmissão. Sua principal função é manter o equilíbrio entre a carga de trabalho e o direcionamento das requisições de uma aplicação, de um site, ou o que estiver em operação no momento, conforme ilustrado na Figura 1.4.

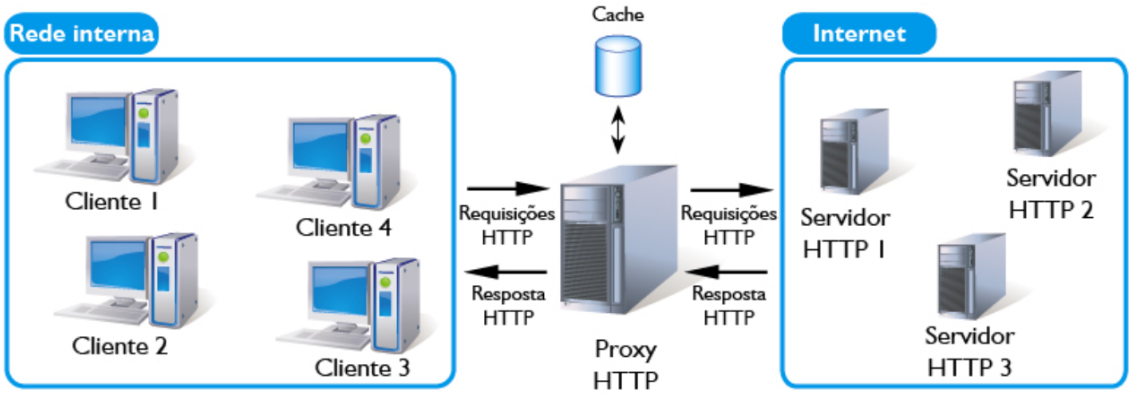

Figura 1.4: Problema de Distribuição de Requisição, Curso de Redes de Computadores UFRN [8].

O Problema de Posicionamento de Réplicas, também pertence à classe de problemas NP-difícil, Aioffi et al. [11] e Zhou et al. [59], consiste em encontrar os melhores servidores dentro da rede sobreposta para alocar os conteúdos replicados, reduzir os custos de transmissão de modo que toda a demanda seja atendida. A resolução do PPR demanda conhecimento das posições dos servidores, bem como sua capacidade de banda e armazenamento, Neves [44].

Segundo De Paula Junior [22], a Figura 1.5 ilustra o PPR, a qual na Figura 1.5(a), tem-se os clientes com suas respectivas requisições de conteúdos e os servidores com os seus conjuntos de réplicas inicialmente vazios. Na Figura 1.5(b), tem-se os posicionamentos das réplicas dos conteúdos em cada servidor, de acordo com as requisições dos clientes e limitações de recursos.

Assim, o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR), ilustrado na Figura 1.6, é uma variante do PPR e consiste em encontrar os melhores locais para posicionar as réplicas dos conteúdos e definir em quais servidores serão atendidas cada uma das requisições a fim de que o tráfego dentro da rede seja minimizado, Neves [44] e Ruiz et al. [36].

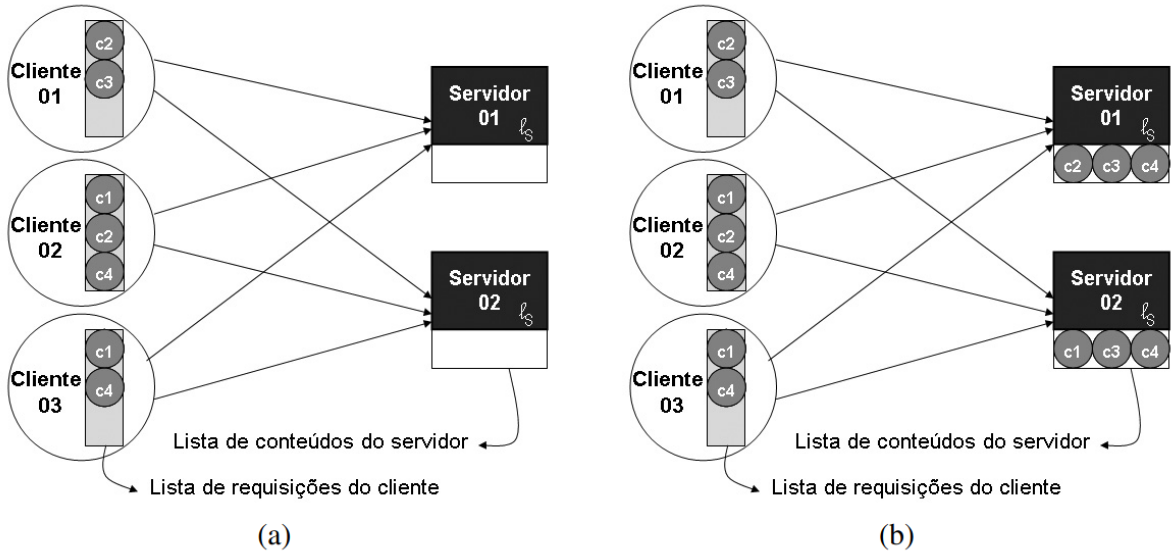

Figura 1.5: Problema de Posicionamento de Replicas, De Paula Junior [22].

Segundo Neves [44], o PPRDR apresenta o posicionamento das réplicas e distribuição eficiente das requisições. O autor relata que o PPRDR ainda é pouco explorado pela comunidade científica, apesar de sua visível importância econômica, fato este que justifica e motiva o desenvolvimento de técnicas eficientes para resolver os problemas decorrentes do tráfego em redes digitais.

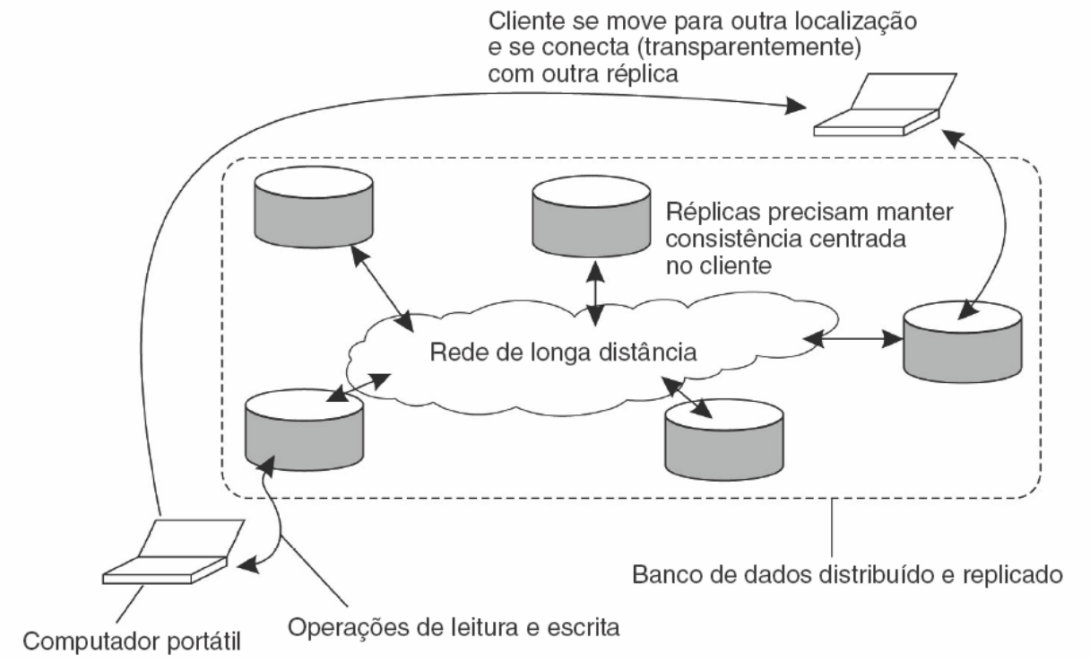

Figura 1.6: Problema de Posicionamento de Réplicas e Distribuição de Requisições, [9].

Para melhor lidar com o surgimento de requisições e conteúdos nas redes digitais, o

horizonte de planejamento é discretizado em períodos de tempo. Requisições com Qualidade de Serviço, em Inglês *Quality of Service* ou QoS, são tipicamente encontradas em ambientes onde existem usuários que pagam por algum tipo de vantagem, como uma melhor qualidade no conteúdo ou uma melhor velocidade de acesso, Garg et al. [26]. Neste sentido, é natural esperar que requisições atendidas por uma CDN, que visam melhorar a experiência do cliente, possuam alguma métrica ou requisito QoS. Em Neves [44] alguns requisitos deste tipo foram abordados no processo de otimização de CDNs.

A redistribuição de requisições é adotada como mecanismo de balanceamento de carga na rede. O objetivo desta redistribuição é reduzir a carga nos servidores e melhorar o fornecimento e acesso aos dados e conteúdos digitais. Em um sistema de redistribuição simples, as requisições são atendidas pelos servidores geograficamente mais próximos, mas, em alguns casos, pode ser vantajoso, adotar um servidor que se encontra um pouco mais distante, mas que não esteja tão carregado quanto o servidor mais próximo, Neves [44].

O posicionamento das réplicas está sujeito a fatores relacionados com os servidores como, por exemplo, capacidade de armazenamento e distâncias entre os mesmos. Devido à possibilidade da existência de custos associados como transporte destas réplicas, a relação custo-benefício entre a redução de carga na rede e o custo de transporte das réplicas deve ser analisada antes da distribuição. Estes fatores tornam o posicionamento das réplicas um problema não trivial, no qual uma decisão precipitada pode acarretar grandes custos, inviabilidades nos requisitos de QoS ou ambos, Li et al. [38].

Além do PPRDR, existe o Problema de Alocação da Capacidade de Armazenamento (PACA), que consiste em determinar de forma ótima qual proporção do espaço total de armazenamento disponível na CDN deve ser alocada sem esgotar a capacidade de armazenamento dos servidores, Uderman et al. [53]. O modelo matemático abordado neste trabalho foi criado por Boechat [18], o Problema de Distribuição de Capacidades, Réplicas e Requisições (PDCRR), o qual contempla a otimização simultânea das capacidades de disco dos servidores, do posicionamento de réplicas e da distribuição de requisições, resolvendo de maneira conjunta dois problemas de otimização que estão presente no contexto das CDNs, o PPRDR e o PACA.

O PDCRR é um problema combinatório e pode ser modelado utilizando Programação Linear Inteira Mista (PIM), para estes problemas a busca por uma solução exata, na maioria das vezes, não são bem sucedidas para os tamanhos de problema de interesse em aplicações do mundo real. Solucionadores exatos de PIM são ferramentas sofisticadas e projetadas para encontrar uma solução ótima comprovável do PIM dentro de um tempo

de computação aceitável.

Mesmo considerando os pacotes de solução de PIM mais atuais, instâncias de tamanho real do PDCRR da formulação matemática apresentada por Boechat [18] não foram resolvidas otimamente em um tempo de computação aceitável.

O *local branching*, técnica proposta por Fischetti [25], é um método de solução exata de PIM. Segundo os autores, o *local branching* controla algum outro algoritmo de solução de PIM, priorizando a exploração parcial ou completa da vizinhança de soluções viáveis já identificadas. As vizinhanças são obtidas por meio da introdução no modelo PIM de desigualdades lineares auxiliares, chamadas cortes locais de ramificação.

Embora seja projetado para melhorar o comportamento heurístico do solucionador PIM disponível, o *local branching* alterna ramificações estratégicas de alto nível para definir vizinhanças de solução e ramificações táticas de baixo nível (realizadas dentro do solucionador PIM) para explorá-los e gradualmente chegar na solução ótima do problema. Um aspecto importante é que o *local branching* melhora o comportamento heurístico do resolvedor sem prejuízo à otimização do método, ou seja, utilizado em conjunto com o resolvedor e dado o tempo de processamento necessário o *local branching* irá encontrar uma solução comprovadamente ótima, Fischetti [25].

O Algoritmo Genético de Chaves Aleatórias Viciadas ou *Biased Random-Key Genetic Algorithm*, BRKGA, é uma meta-heurística recente proposta por Gonçalves et al [30]. Segundo Gendreau et al. [28], métodos meta-heurísticos são métodos que coordenam outros métodos para a busca de soluções ótimas locais e com estratégias de alto nível para fugir desses ótimos locais de baixa qualidade, dessa forma, possibilita ter uma maior chance de se encontrar uma solução ótima global ou mais próxima dela, sem haver a necessidade de explorar todo o espaço de soluções de um problema.

A proposta presente nos algoritmos genéticos consiste em estabelecer uma população inicial de "cromossomos" ou indivíduos. A cada geração, um grupo dos cromossomos é escolhido para se reproduzir e frequentemente os mais aptos, os de melhor valor da função objetivo, produzem mais descendentes do que os demais, desta forma, a cada geração o algoritmo melhora a qualidade da função objetivo.

Neste sentido, este trabalho visa explorar o uso das técnicas de *local branching* e de algoritmos genéticos de chaves aleatórias viciadas para a resolução do PDCRR. Os objetivos do trabalho estão descritos detalhadamente a seguir.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral do presente trabalho é realizar a implementação e análise do método *Local Branching* Clássico no modelo *offline* no PDCRR, bem como a aplicação do Algoritmo Genético de Chaves Aleatórias Viciadas no mesmo problema.

1.3.2 Objetivos específicos

Aplicar e analisar o Algoritmo *Local Branching* Clássico ao PDCRR em Redes de Dados Distribuídos para grandes instâncias do problema.

Aplicar e analisar Algoritmo Genético de Chaves Aleatórias Viciadas ao PDCRR em Redes de Dados Distribuídos para grandes instâncias do problema.

Comparar os resultados obtidos entre os métodos propostos e os resultados do modelo exato. O modelo exato do PDCRR foi estabelecido por Neves [44] e aprimorado por Boechat [18].

O restante deste trabalho está organizado da seguinte maneira:

- Capítulo 2: descreve-se de forma detalhada o modelo matemático do PDCRR;
- Capítulo 3: descreve a técnica do *local branching*, proposta por Fischetti [25], e apresentar algumas heurísticas voltadas para a implementação do algoritmo;
- Capítulo 4: apresenta o Algoritmo Genéticos de Chaves Aleatórias Viciadas, uma meta-heurística recente proposta por Gonçalves et al. [30];
- Capítulo 5: apresenta as característica do ambiente onde foram realizados os testes computacionais, bem como a forma como foram geradas as instâncias para o problema e suas principais características, expondo os resultados obtidos, bem como uma análise dos mesmos; e
- Capítulo 6: encontram-se as conclusões do trabalho e algumas propostas para trabalhos futuros.

Capítulo 2

MODELO MATEMÁTICO

O modelo matemático que será empregado neste trabalho foi criado por Boechat [18] e contempla a otimização simultânea das capacidades de disco dos servidores, do posicionamento de réplicas e da distribuição de requisições, resolvendo de maneira conjunta dois problemas de otimização que estão presentes no contexto das CDNs, o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR) e o Problema de Alocação da Capacidade de Armazenamento (PACA).

2.1 Problema de Alocação da Capacidade de Armazenamento

Segundo Boechat [18], o PACA tem influência direta no PPRDR, na qual o PPRDR consiste em determinar o posicionamento ótimo das réplicas dos conteúdos na rede e distribuir as requisições pelos servidores, a fim de reduzir o custo de funcionamento da rede, atendendo aos critérios de qualidade. Ao se fazer uma réplica, o conteúdo é copiado integralmente no outro servidor, tornando-se necessário que a capacidade de armazenamento destes servidores sejam avaliadas e a alocação desta réplica seja feita de forma adequada, tornando relevante a restrição de espaço.

Para tentar resolver o PACA, um estudo feito por Uderman et al. [53] analisou uma série de problemas matemáticos que tratam separadamente estes conceitos e montou um modelo matemático para a solução do PACA adequada para CDNs sem uma topologia hierárquica definida e capaz de suportar conteúdos de tamanhos não unitário, reduzindo o número de variáveis necessárias para descrever o modelo.

De acordo com Uderman et al. [53], o PACA consiste em determinar de forma ótima

a distribuição da capacidade total de armazenamento disponível entre os servidores da CDN. Neste sentido, servidores próximos à concentração de pontos de demanda por um determinado conteúdo devem possuir uma réplica e portanto, deve ser alocado espaço suficiente para acomodar a referida réplica nestes servidores. Contudo, o volume total de espaço que pode ser distribuído entre os múltiplos servidores da rede é limitado e deve ser observado o custo-benefício desta alocação.

2.2 Problema de Posicionamneto de Réplicas e Distribuição de Requisições

Com relação ao PPRDR, segundo Neves [44], para distribuir as requisições, o PPRDR considera que esta só será redirecionada para um servidor que possua uma cópia do conteúdo solicitado, averiguando as restrições de qualidade que, se violadas, devem sofrer uma penalização no modelo. Desta maneira, em certas situações pode ser necessário alterar o número de réplicas e alterar a distribuição. Já que o objetivo do PPRDR é melhorar a qualidade de serviço, em alguns casos pode ser necessário usar um servidor mais distante, porém menos carregado.

Uma formulação do PPRDR proposta por Neves [44] considera as demandas divisíveis, não havendo diferença entre os custos de replicação de um conteúdo. Além disso, trabalha com conteúdos múltiplos, considerando que cada demanda é associada a um conteúdo. Esta formulação faz uso da técnica de *backlog*, que obriga o atendimento de alguma das partes da demanda nos períodos subsequentes caso a demanda não seja completamente atendida no período corrente. Considera múltiplas origens para um conteúdo, porém cada conteúdo pode ter apenas uma origem, e de atualização, baseando a abordagem em tempo de vida para os conteúdos [28].

Assim, de forma simplificada, o PPRDR pode ser definido da seguinte forma: dado um conjunto R de requisições dos clientes a serem atendidas; um conjunto S de servidores da CDN e C o conjunto de conteúdos replicados, deve-se determinar a melhor localização dessas réplicas nos servidores e a distribuição das requisições pelo servidores de forma a minimizar as penalidades por atraso nas entregas dos conteúdos aos clientes e respeitando os limites de banda máxima e espaço disponível do servidor e da requisição.

O problema foi abordado por Neves [44] como uma variante do Problema de Posicionamento de Réplicas, na qual busca encontrar o melhor posicionamento para as réplicas dentro de uma rede sobreposta e redistribuir as requisições entre os servidores. Para

melhor lidar com a natureza dinâmica do problemas, o horizonte do planejamento foi discretizado em período de tempos correspondentes à 60 segundos e como restrições de QoS foram consideradas duas exigências: banda mínima e atraso máximo.

Como forma de reduzir a carga nos servidores e melhorar a qualidade do serviço, Neves [44] realizou uma redistribuição de requisições. Para realizar esta redistribuição considerou-se os seguintes fatores: uma requisição pode ser redirecionada para um servidor somente se este servidor possui uma réplica do conteúdo; as restrições de QoS da requisição precisam ser averiguadas; se as restrições de QoS forem violadas, a solução produzida é considerada de baixa qualidade e deve ser penalizada; em algumas situações, para poder otimizar a distribuição de requisições, pode ser necessário alterar o posicionamento e número das réplicas; o posicionamento das réplicas está sujeito a capacidade de armazenamento e distâncias entre elas; inicialmente os conteúdos se encontram em seus servidores de origem, podendo um ou mais conteúdos ter o mesmo servidor de origem, e só depois podem ser distribuídos pela rede; a relação custo-benefício entre a redução de carga na rede e o custo de transporte das réplicas deve ser analisada antes da distribuição.

A formulação apresentada por Neves [44] possui as seguintes características:

- A função objetivo minimiza o custo de entrega dos conteúdos para os clientes bem como a quantidade de *backlog* feitos ao longo do tempo e o custo de replicação.
- Associa a fração do conteúdo solicitado pela requisição e o atraso na entrega de conteúdos que será atendida no período posterior. A associação proposta permite que a entrega de uma fração do conteúdo possa ser postergada (enviada em um período posterior ao solicitado) desde que uma penalidade seja paga por este atraso.
- Exige que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima.
- Impede que seja entregue ao cliente uma banda maior do que ele suporta.
- Condiciona que uma requisição só pode ser atendida por servidores que possuam uma réplica do conteúdo exigido.
- Controla o número de réplicas de um conteúdo, afirmando que no mínimo uma réplica deve existir durante o tempo de vida do conteúdo e que nenhuma réplica pode existir fora do tempo de vida.
- Faz com que, no período de submissão, apenas o servidor origem de um conteúdo possua uma réplica.

- Garante que réplicas não surjam espontaneamente.
- Exige que uma replicação ocorra a partir de um servidor que possua o conteúdo replicado.
- Diz que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível.

2.3 Problema de Distribuição de Capacidades, Réplicas e Requisições

Nas análises realizadas por Boechat [18], o PDCRR pode ser visto como um problema mais geral, que engloba o PPRDR e o PACA. Afim de evitar maiores dificuldades para inclusão de todos os conceitos do PPRDR na formulação do PACA, torna-se mais coerente realizar modificações na formulação de Neves [44] para contemplar os aspectos faltantes e criar uma nova formulação que trate de forma mais geral todos estes conceitos. De acordo com Boechat [17] foram detectadas algumas restrições redundantes que foram removidas visando possibilitar que a formulação pudesse tratar instâncias maiores do problema, já que um dos problemas encontrados em Neves [44] foi a alta demanda por memória.

A formulação proposta por Neves [44] considera as capacidades de cada servidor como constantes, desta maneira, não contempla a análise do espaço em disco utilizado por cada servidor.

Ainda nas análises feitas por Boechat [18], em redes que os clientes exigem mais qualidade de serviço, com exigências de banda e atraso, a capacidade dos servidores pode acabar sendo exaurida tornando assim importante o estudo do PACA. Além disso, as capacidades de armazenamento dos servidores influenciam na resolução do PPRDR, uma vez que estas capacidades podem ser consideradas parâmetros para o gerenciamento de réplicas e requisições.

Uma vez identificadas e definidas à interdependência entre o PACA e o PPRDR, ambos os problemas foram estudados por Uderman et al. [53], a fim de que os resultados do primeiro fossem diretamente utilizados como dados de entrada para o segundo. Nesta abordagem, o PACA é utilizado para encontrar uma distribuição otimizada do espaço de armazenamento para cada servidor de CDN. No estudo de Uderman et al. [53] utiliza duas abordagens. Na primeira o PACA é resolvido utilizando taxas médias das requisições, porém os resultados obtidos pela resolução do PACA são utilizados apenas uma

vez, no início da criação da CDN. Na segunda, o PACA é resolvido para cada período do horizonte de planejamento, considerando como dados de entrada o estado atual da CDN. Entretanto, esta segunda abordagem continua tratando os dois problemas (PACA e PPRDR) de maneira isolada, o que faz com que as mudanças ocorridas ao longo do tempo não sejam observadas com o devido cuidado.

Neste sentido, Boechat [18] propõe um novo problema de otimização, nomeado Problema de Distribuição de Capacidades, Réplicas e Requisições (PDCRR), em que o objetivo principal é a alocação ótima da capacidade de armazenamento disponível, distribuição das réplicas de conteúdos e das requisições de clientes nos servidores, afim de otimizar a utilização dos recursos e reduzir os custos operacionais, mantendo os padrões de qualidade de serviço (QoS). Deste modo, o PDCRR pode ser visto como um problema mais geral, que engloba o PPRDR e o PACA.

Para construir uma formulação matemática para o PDCRR Boechat [18] usa a formulação exposta em Neves [44], por meio da inclusão de algumas restrições que possibilitam um formulação que também consegue lidar com a otimização do espaço de armazenamento sob a forma de um Problema de Programação Linear (PPL), com as alterações necessárias para incluir os conceitos do PACA para resolver o problema de maneira conjunta.

O PPL apresentado como método de resolução do PDCRR é utilizado para versões *offline* do mesmo, ressaltando que o estudo de problemas *offline* é interessante pois serve como mecanismo para percepção do comportamento da rede, o que permite analisar estratégias de alocação e distribuição mais eficiente para os problemas *online*.

Segundo Bazaraa [12], um Problema de Programação Linear é um problema de maximização ou minimização de uma função linear na qual esta função está sujeita à um conjunto de restrições também lineares (2.2)-(2.6), ou seja, dada uma função linear $f(x)$, é possível escrevê-la como $f(x) = c_1x_1 + c_2x_2 + c_3x_3 + \ldots + c_nx_n$, com $c_n \in R$, cujas

restrições são lineares. Veja o exemplo:

$$Min \qquad c_1x_1 + c_2x_2 + c_3x_3 + \ldots + c_nx_n \tag{2.1}$$

$$S.a.$$

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n \geq b_1 \tag{2.2}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n \geq b_2 \tag{2.3}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n \geq b_3 \tag{2.4}$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ldots + a_{\mathrm{mn}}x_n \geq b_m \tag{2.5}$$

$$x_1, \qquad x_2, \quad x_3, \quad \ldots \quad , \quad x_n \geq 0 \tag{2.6}$$

Onde $c_1x_1+c_2x_2+c_3x_3+\ldots+c_nx_n$ é conhecida como função objetivo a ser minimizada e na literatura costuma ser denotada por z. O coeficientes $c_1, \quad c_2, \quad c_3, \quad \ldots \quad , c_n$ são coeficientes de custos, estes custos são conhecidos, e $x_1, \quad x_2, \quad x_3, \ldots, x_n$ são chamadas de variáveis de decisão, nas quais serão determinadas. As inequações são as restrições do problema que limitam a região de solução da função objetivo e os coeficientes a_{ij}, com $i = 1, \quad 2, \quad 3, \quad \ldots, \quad m$ e $j = 1, \quad 2, \quad 3, \quad \ldots, \quad n$ são chamados de coeficientes tecnológicos. Estes coeficientes formam a matriz de restrição A dada por:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \ddots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{\mathrm{mn}} \end{bmatrix}$$

O vetor coluna cujo i-ésimo componente é b_i, referido como vetor do lado direito, representa os requisitos mínimos a serem satisfeitos. As restrições $x_1, \quad x_2, \quad x_3, \ldots, x_n \geq 0$ são restrições de não negatividade. Um conjunto de valores para as variáveis $x_1, \quad x_2, \quad x_3, \ldots, x_n$ que satisfaçam todas as restrições é chamado de ponto ou solução viável e o conjunto de todos esses pontos constitui a região ou espaço viável.

Utilizando as terminologias acima, dizemos que o problema de programação linear pode ser declarado da seguinte forma: Entre todos os vetores viáveis, ou seja, vetores que satisfaçam todas as restrições do problema, encontre aquele que minimiza (ou maximiza) a função objetivo.

Sucintamente pode-se escrever um problema de programação linear da seguinte forma:

$$\begin{aligned} & \min c'x \\ s.a & \\ & \mathrm{Ax} \leq b \\ & x \geq 0 \end{aligned}$$

Quando todas as suas variáveis são inteiras, o modelo é denominado Programação Inteira Pura, caso contrário, é denominado Programação Inteira Mista (PIM).

Ambas as formulações proposta por Neves [44] e Boechat [18] usam PIM como abordagem exata dos problemas e têm as seguinte definições matemáticas: Consideram-se R o conjunto de requisições a serem atendidas, S o conjunto de servidores da CDN, C o conjunto de conteúdos a serem replicados e T o conjunto de períodos de tempo. Sejam também as constantes $\delta = 60$ a duração do período (em segundos), origem (i) o servidor de origem da requisção i, $G(i)$ o conteúdo exigido pela requisicão i, ld(i) o atraso local da requisição i (em segundos), BR_i a banda mínima da requisição i (em bytes/segundo), BX_i banda máxima da requisição i (em bytes/segundo), TD_i o atraso máximo permitido pela requisição i. Além destas definições também utiliza-se L_k o tamanho do conteúdo k (em bytes) B_k período em que o conteúdo k é disponibilizado, E_k período em que o conteúdo k é removido da CDN, O_k o servidor origem do conteúdo k, AS_j o espaço em disco disponível no servidor j (em bytes) e MB_j a banda máxima do servidor j (em bytes/segundo). Sejam ainda atraso(j, l, t) o atraso de comunicação (em segundos) partindo do servidor j para o servidor l no período t e RTT(j, j, t) o tempo que uma mensagem (ou pacote) leva para percorrer o caminho de um servidor j para um outro servidor l e voltar ao primeiro em um período t dado por $\mathrm{RTT}\,(j, l, t) = \mathrm{atraso}\,(j, l, t) + \mathrm{atraso}(l, j, t)$ (expresso em segundos).

A formulação matemática utiliza as seguintes variáveis: x_{ijt} variável contínua que representa a fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t; A variável binária y_{kjt} assume 1 se o conteúdo k está replicado no servidor j no período t, e 0 caso contrário; A variável contínua b_{it} é o *backlog* da requisição i no período t, ou seja, é o montante da demanda de i que deveria ser atendido em t mas que é postergado para algum período seguinte; E a variável binária w_{kjlt} que assume 1 se o conteúdo k é copiado pelo servidor j a partir do servidor l no período t, e 0 caso contrário.

Além das constantes já definidas, as seguintes definições são utilizadas: D_{it} é a demanda da requisição i no período t (em bytes); c_{ijt} é o custo de atender a requisição

i pelo servidor j, no período t, dado por $c_{\mathrm{ijt}} = (\omega + \mathrm{RTT}\left(\mathrm{origin}\left(i\right), j, t\right)) \times BR_i$ se $\omega \leq TD_i$, senão $c_{\mathrm{ijt}} = ((\omega + \mathrm{RTT}\left(\mathrm{origin}\left(i\right), j, t\right)) \times BR_i + 1000 \times (\omega - TD_i) + 1000$ onde $\omega = \mathrm{atraso}\left(\mathrm{origin}\left(i\right), j, t\right) + \mathrm{ld}(i)$; p_{it} penalidade por fazer *backlog* da requisição i no período t dada por $p_{\mathrm{it}} = \max\left(c_{\mathrm{ijt}}\right) \times 2, \forall\ i \in R,\ \forall\ t \in T$; E h_{kjlt} custo de replicar o conteúdo k no servidor j a partir do servidor l no período t. Nos experimentos feitos, este custo é sempre dado pelo tamanho do conteúdo k, ou seja, $h_{\mathrm{kjlt}} = L_k,\ \forall\ k \in C$.

De forma resumida, dado um conjunto R de requisições dos clientes a serem atendidas; um conjunto S de servidores CDN e C conjunto de conteúdos replicados, deve-se determinar a melhor localização dessas réplicas no servidores e a redistribuição das requisições pelo servidor de origem de forma a minimizar as penalidades p_{it} por atraso nas entregas dos conteúdos k aos clientes e respeitando aos limites de banda máximo e espaço disponível no servidor e da requisição.

No trabalho de Boechat [18] foi observado que as capacidades de cada servidor são constantes no PPRDR, porém, para o PDCRR, a capacidade de cada servidor passa a ser variável. A formulação proposta Boechat [18] utiliza variáveis r_{jt} que representam o espaço em disco alocado no servidor j, durante o período t, ao invés das constantes AS_j da formulação proposta no trabalho Neves [44]. Uma versão particular desta formulação utiliza variáveis r_j, para o caso onde a alocação de espaço nos servidores deve ser homogênea ao longo do tempo. No caso homogêneo, a capacidade de cada servidor deve assumir o mesmo valor em todo o horizonte de planejamento. Portanto, para incluir os conceitos do PACA no PPRDR, Boechat [18] realizou as seguintes alterações na formulação feita for Neves [44]:

- Retirou as restrições que garantiam que a capacidade de disco dos servidores não seria violada, devido a redundância de restrições;
- Incluiu as restrições para o problema onde a alocação deve ser homogênea no tempo ou para o caso onde a alocação pode mudar com o tempo;

Com as definições expostas, a formulação proposta por Boechat [18] assume a seguinte forma:

Considerando S o conjunto de servidores da CDN, C o conjunto de conteúdos replicados, R o conjunto de requisições a serem atendidas e E o espaço total disponível para distribuição entre os servidores. Sejam as contantes c_{ijt} custo de atendimento da requisição i no servidor j, no período t, q_{it} penalidade por usar *backlog* da requisição i no período

t, h_{kjlt} o custo de replicar o conteúdo k no servidor j a partir do servidor l no período t. Além disso, L_k o tamanho do conteúdo k (em *bytes*), MB_j banda máxima do servidor j (em *bytes*/segundo), BR_i exigência de banda da requisição i (em bytes/segundo), $G(i)$ o conteúdo exigido pela requisição i e M uma constante que representa a soma do tamanho de todos os conteúdos.

Define-se também os seguintes conjuntos de variáveis:

- x_{ijt} = fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t
- $y_{\text{kjt}} = \begin{cases} 1, & \text{se e somente se o conteudo } k \text{ esta replicado no servidor } j \text{ no periodo t} \\ 0, & \text{, caso contratrio} \end{cases}$
- b_{it} = backlog da requisição i no período t
- $w_{\text{kjlt}} = \begin{cases} 1, & \text{se } e \text{ somente se } o \text{ conte}u\text{do } k \text{ } e \text{ copiado pelo servidor } j \text{ } a \\ & \text{partir do servidor } l \text{ no per}i\text{odo } t \\ 0, & \text{caso contratrio} \end{cases}$

  Constantes:
- c_{ijt} custo de atendimento da requisição i no servidor j, no período t.
- q_{it} penalidade por usar *backlog* da requisição i no período t.
- h_{kjlt} o custo de replicar o conteúdo k no servidor j a partir do servidor l no período t.
- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da CDN.
- C conjunto de conteúdos replicados.
- T conjunto de todos os períodos.
- L_k o tamanho do conteúdo k (em *bytes*).
- MB_j banda máxima do servidor j (em *bytes*/segundo).
- BR_i exigência de banda da requisição i (em *bytes*/segundo).
- $G(i)$o conteúdo exigido pela requisição i.

- M uma constante que representa a soma do tamanho de todos os conteúdos.

A formulação matemática é dada por:

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} + \sum_{i\in R}\sum_{t\in T} q_{it}b_{it} + \sum_{k\in C}\sum_{j\in S}\sum_{l\in S}\sum_{t\in T} h_{kjlt}w_{kjlt} + F\sum_{j\in S}\sum_{t\in T} r_{jt} \tag{2.7}$$

$$S.a.$$

$$\sum_{j\in S} L_{G(i)}x_{ijt} - b_{i(t-1)} + b_{it} = D_{it}\ ,\ \forall i \in R,\ \forall t \in [B_{G(i)}, E_{G(i)}], \tag{2.8}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq \delta M B_j\ ,\ \forall j \in S,\ \forall t \in T, \tag{2.9}$$

$$\sum_{j\in S} L_{G(i)}x_{ijt} \leq \delta B X_i\ ,\ \forall i \in R,\ \forall t \in T, \tag{2.10}$$

$$y_{G(i)jt} \geq x_{ijt}\ ,\ \forall i \in R,\ \forall j \in S,\ \forall t \in T, \tag{2.11}$$

$$\sum_{j\in S} y_{kjt} \geq 1\ ,\ \forall k \in C,\ \forall t \in [B_k, E_k]\,, \tag{2.12}$$

$$y_{kjt} = 0\ ,\ \forall k \in C,\ \forall j \in S,\ \forall t \notin\ [B_k, E_k]\,, \tag{2.13}$$

$$y_{kjB_k} = 0\ ,\ \forall k \in C,\ \forall j \in \{S|_j \neq O_k\} \tag{2.14}$$

$$y_{kj(t+1)} \leq \sum_{l\in S} w_{kjlt}\ ,\ \forall k \in C,\ \forall j \in S,\ \forall t \in T, \tag{2.15}$$

$$y_{kjt} \geq w_{kljt}\ ,\ \forall k \in C,\ \forall j \in S, \forall l \in S,\ \forall t \in T, \tag{2.16}$$

$$\sum_{k\in C} L_k y_{kjt} \leq r_{jt}\ ,\ \forall j \in S,\ \forall t \in T, \tag{2.17}$$

$$\sum_{j\in S} r_{jt} \leq E\ ,\ \forall t \in T, \tag{2.18}$$

$$x_{ijt} \in [0,1]\,,\ \forall i \in R,\ \forall j \in S,\ \forall t \in T, \tag{2.19}$$

$$y_{kjt} \in \{0,1\},\ \forall k \in C,\ \forall j \in S,\ \forall t \in T, \tag{2.20}$$

$$b_{it} \geq 0,\ \forall i \in R,\ \forall t \in T, \tag{2.21}$$

$$w_{kjlt} \in \{0,1\}\,,\ \forall k \in C,\ \forall j \in S, \forall l \in S,\ \forall t \in T, \tag{2.22}$$

$$0 \leq r_{jt} \leq M,\ \forall j \in S,\ \forall t \in T. \tag{2.23}$$

Desta forma a formulação propostas por Boechat [18] pode ser descrita como segue:

A função objetivo 2.7: Minimiza o custo de entrega dos conteúdos aos clientes bem como a penalidade por atraso de atendimento das requisições e ainda o custo de replicação dos conteúdos nos servidores. A constante F representa o custo de alocação de conteúdo

em um servidor, considerado fixo para toda a rede.

Restrições 2.8: Garantem que a demanda será totalmente atendida, fazendo um controle que indica que a quantidade entregue do período atual somada à penalidade do período posterior deverá ser equivalente à demanda do período atual somada às penalidades por atraso do período anterior.

Restrições 2.9 e 2.10: Limitam a quantidade de conteúdos entregues à banda máxima permitida pelo servidor e as exigências de banda do cliente.

Restrições 2.11: Garantem que uma requisição só pode ser atendida por um servidor que possua uma cópia do conteúdo solicitado.

Restrições 2.12 e 2.13: Controlam as réplicas, garantindo que haja pelo menos uma cópia do conteúdo durante seu período de existência e que não tenha nenhuma réplica fora deste período.

Restrições 2.14: Garantem que durante o surgimento do conteúdo, todos os outros servidores, exceto o servidor origem, não contenham nenhuma réplica do conteúdo.

Restrições 2.15: Exigem que seja criada uma nova réplica a cada replicação.

Restrições 2.16: Exigem que a replicação deve ocorrer a partir de um servidor que possua o conteúdo replicado.

Restrições 2.17: Indica que a soma do espaço ocupado em um servidor não deve ultrapassar o espaço alocado neste servidor.

Restrições 2.18: Garantem que, em cada período, a soma dos espaços em discos alocados nos servidores deve ser menor ou igual que o espaço total disponível.

As demais restrições são de integralidade e não negatividade.

2.4 Trabalhos Relacionados

Na literatura o trabalho mais completo encontrado e que trata de problemas em CDN de forma mais abrangente e completa é o de Neves [44], o PPRDR, Problema de Posicionamento de Réplicas e Distribuição de Requisições, e posteriormente Boechat [18] acrescenta o PACA ao PPRDR e cria o PDCRR, Problema de Distribuição de Capacidades, Réplicas e Requisições, trabalhos estes já citados anteriormente. Este trabalho apresenta dois métodos na busca por melhores soluções para o PDCRR. O que encontramos na literatura são diversos trabalhos que tratam de forma isolada os problemas dentro de uma CDN,

Problema de Localização de Servidores (PLS), Problema de Replicação (PR), Problema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR), Problema de Distribuição de Capacidades e Réplicas, (PDCR), Problema da Alocação de Capacidade de Armazenamento, PACA, Problema de Fluxo de Custo Mínimo, PFCM. Serão apresentados alguns destes problemas a seguir, bem como a abordagem utilizada pelos autores para a busca da solução dos mesmos:

Com o aumento de volume e popularidade dos conteúdos, as estruturas de CDNs por si só podem se tornar inadequadas. Neste contexto, uma arquitetura de CloudCDN, Hu et al. [34], pode apresentar novas perspectivas. Uma CloudCDN é construída numa infraestrutura de fornecimento de capacidade em nuvem. Conforme Hu et al. [34], comparando com uma estrutura tradicional de CDN, este tipo de arquitetura em nuvem pode prover dois benefícios: personalização da estrutura sem alto custo de aquisição ou operação de servidores geograficamente dispersos; pequenas companhias tem possibilidade de alugar um serviço de CDN em nuvem de um fornecedor de armazenamento em nuvem, utilizando-se de um tipo de pagamento por demanda, reduzindo custos.

O trabalho de Chen et al. [20], discursa sobre a construção de CDNs sobre estruturas em nuvem e Wu et al. [57] tratam de padrões de demanda e acesso de usuários em aplicações tradicionais de video por demanda (VoD), além de desenvolver serviços em plataformas em nuvem. As CDNs em nuvem são capazes de prover redução no custo operacional e eficiência na distribuição dos conteúdos, Hu et al. [34].

Além disso, em Wu et al.[56], os autores tratam um tipo de CDN acadêmica, conhecida omo CoDeeN, onde os clientes configuram seus navegadores a usar servidor CoDeeN, que atua como um *proxy forward*. Assim, erros de cache são desordenados e redireionados para outro *proxy* CoDeeN que atua de modo inverso, concentrando pedidos de um provedor particular. Deste modo, menos solicitações são encaminhadas ao provedor origem. O CoDeeN não opera em servidores dedicados e com recursos confiáveis, nem emprega um centro de operações de rede para coletar e distribuir informações de status. Cada instância monitora a saúde do sistema e fornece os dados de redirecionamento de requisições locais.

Com a utilização da alocação dinâmica de capacidades, surgem também os estudos sobre virtualização e otimização das capacidades. Em Lin et al. [40] ocorre a implementação de servidores de uma CDN em máquinas virtuais que podem ter seus recursos alocados dinamicamente.

O trabalho de Bjorkqvist et al. [16] afirma que a capacidade total disponível para o serviço da CDN que pode ser realocada entre os servidores conforme as taxas de requisição

por diferentes conteúdos oscila em uma CDN. Assim, evita a subutilização dos servidores e torna possível o atendimento da demanda cada vez mais crescente por conteúdos.

Os autores Li et al. [39] tratam de análises em topologias hierárquicas, estudando a ideia do *cache* corporativo, em que alguns nós devem ter os conteúdos carregados na memória. Desta maneira, um conteúdo pode ser enviado de um servidor vizinho sem percorrer o caminho até o servidor central, acelerando o tempo de resposta e reduzindo o custo de comunicação, além de reduzir a carga de conteúdos nos servidores. Por questões de restrições de carga, é impossível que todos os servidores armazenem todos os conteúdos do sistema, logo, tem-se por objetivo decidir em qual servidor armazenar um conteúdo para alcançar a máxima eficiência do sistema.

Yuqiang et al. [55] tratam o sistema CDN tradicional como um caso especial de sistema CDN hierárquico. No sistema CDN tradicional e no sistema hierárquico CDN, analisando a largura de banda entre os subsistemas, encontra-se o que os autores chamam de inter-sistema largura de banda, apontado como um gargalo que impede a expansão do sistema CDN. Para resolver este problema, os autores propõe um novo tipo de arquitetura de sistema distribuído que usa canais de transmissão para distribuir dados do tipo *broadcast* e ainda usar canais *unicast* bidirecionais para outros tipos de dados, denominando a nova arquitetura como CHCDN (*Channel Heterogeneous CDN*). A nova arquitetura foi analisada e comparada com a arquitetura tradicional do sistema CDN e a arquitetura hierárquica do sistema CDN. O resultado da simulação experimental mostrou que o sistema CHCDN apresenta melhor desempenho em atualização de cache de tempo e recursos com maior eficiência de transmissão de dados do que o sistema CDN hierárquico, o que indica que a nova arquitetura tem grande potencial para ser amplamente utilizada em computação distribuída.

O trabalho de Samaneh et al. [35] realiza uma abordagem sobre CDNs voltado para um estudo sobre o consumo de energia em redes, a qual cita que a maior parte da energia é consumida pelos pontos de cache (CPs) e o equipamento associado, visando implantar menor número de CPs ou o desligamento do maior número possível para economizar energia. Isso resulta em qualidade degradada de serviço (QoS). Os autores utilizam uma técnica usual de dimensionamento para otimizar o número e as localizações dos pontos de cache (CPs) de uma rede de distribuição de conteúdo (CDN), onde o objetivo é reduzir a energia operacional. O trabalho utiliza métodos heurísticos para provisionamento de alto número de *downloads* simultâneos, o que reduz tempo total de espera e número de pedidos descartados de usuários de veículos da cidade. O resultado final é substancial

melhoria da qualidade de serviço (QoS). O proposto CPs economizam 100% de energia da rede durante todo o dia enquanto atendendo a demanda de conteúdo.

O artigo de Zhenyou et al. [47] apresenta uma nova abordagem para armazenamento em cache em CDNs que usa aprendizado de máquina, para realizar esta tarefa complexa, propõe um design de cache CDN chamado *Learning Relaxed Belady* (LRB) para imitar um algoritmo *Relaxed Belady*, usando o conceito de Limite de *Belady*. O documento aborda vários desafios para construir um protótipo de cache de aprendizado de máquina, incluindo como reunir dados de treinamento, limitar a sobrecarga de memória e ter treinamento e previsão. Implementaram um simulador de LRB e um protótipo dentro do *Apache Traffic Server*, servidor *web*.

Bektas et al. [14] tratam vários problemas relacionados à CDN, incluindo o Problema de Dimensionamento e Posicionamento de Servidores e o PPRDR', estáticos. O PPRDR' é similar ao PPRDR sendo a principal diferença destes o fato de que a QoS não é tratada no primeiro. Um modelo matemático não linear é proposto, bem como uma linearização para o mesmo. Os autores também apresentam um algoritmo exato baseado na decomposição de Benders, Maniezzo et al. [41]. Um algoritmo heurístico guloso também é proposto. Este algoritmo ativa um servidor a cada iteração seguindo uma ordem de prioridade predeterminada. Esta ordem de prioridade pode ser, por exemplo, a ordem crescente do custo de ativação dos servidores ou ainda a ordem crescente de capacidade de armazenamento dos mesmos. Os resultados computacionais mostraram que os algoritmos propostos atingiram boa performance, alcançando uma diferença de no máximo 5% em relação à solução ótima.

No artigo de Tang et al. [51] trabalham com uma versão do PPRDR para um único conteúdo em sua versão estática. O modelo apresentado também tenta prover QoS através da redução da distância entre o servidor que faz o atendimento e o ponto de origem da requisição. Assim como em outros trabalhos, a noção de distância é generalizada para se encaixar em diversos tipos de métricas. Vários algoritmos gulosos simples são propostos e validados experimentalmente através de comparação com uma relaxação linear do problema. Os autores afirmam que heurísticas gulosas são capazes de produzir soluções próximas do ótimo com baixo esforço computacional para o problema abordado.

Os autores Manoj et al. [49] afirmam, em seu trabalho, que as grandes velocidades no uso de redes de computadores, variam desde instalações, gestão eficiente de conteúdos, disponibilidade de recursos, entre outros. Afirmam que estas atividades tendem a aumentar, sendo primordial que o sistema seja seguro e rápido quanto à sua utilização. Os

autores afirmam que a entrega de conteúdo depende da localização do Servidor de Origem e do Servidor de Cache, e descrevem uma matriz matemática usando o conceito de Teoria do Transporte para mostrar a otimização entre o Servidor de Cache e o Cliente durante a entrega de conteúdo em CDNs e utilizam lógica *fuzzy* para otimizar o gerenciamento de entrega de conteúdo na CDN. Os autores usam *machine learning* para resolver o PPR, fazendo uma integração de múltiplos servidores cache para lógica *fuzzy* na CDN, a qual a seleção baseada em conteúdos para vários servidores de cache que processam e executam solicitações da mesma forma, no entanto, a distância entre o usuário e servidor desempenha um papel importante. No artigo, Manoj et al. [49], consideram uma abordagem condicional na execução pelo servidor, estando o usuário conectado a um servidor cache, o algoritmo leva em consideração primeiramente a distância entre os servidores cache, em seguida é feita a escolha da menor distância entre os servidores e o usuário, uma vez encontrado o caminho mínimo, o usuário será atendido por este servidor. Para obter uma solução ótima com vários usuários, são considerados os valores das distâncias mínimas entre o servidor central para os outros servidores e os valores dos tempos referentes à essas distâncias mínimas. Manoj et al. [49] cita Venketesh et al. [54] e descrevem que, se o tempo for constante, o custo será diretamente proporcional ao valor da distância, se a distância for constante, o custo será diretamente proporcional ao tempo, e quando o tempo e a distância não são constantes, o custo será proporcional a uma constante de variação multiplicada pelo produto da distância e do tempo. Considerando estes conceitos, o cache disponível para diferentes usuários de vários domínios reconheceria a rota armazenada nos roteadores para um conteúdo já requisitado anteriormente e esta rota prevaleceria para novas requisições, Berger et al. [15]. Manoj et al. [49] concluem que a taxa média de download dependeria subsequentemente do cache disponível que reside nos roteadores da rede e que a entrega de conteúdo depende da localização do Servidor de Origem e do Servidor de Cache.

No trabalho de Aioffi et al. [11], abordam uma versão dinâmica do PPRDR', sendo contextualizada em um ambiente de redes móveis corporativas. Neste trabalho são apresentadas uma formulação matemática, usada para o modelo *offline* e uma heurística, baseada em um método de previsão, usada para a versão *online* do problema. A heurística utiliza um algoritmo de previsão de demanda para cada um dos servidores. Baseados nestas previsões, os servidores calculam os custos de manter conteúdos replicados, replicar novos conteúdos e também de atender remotamente as demandas. A partir destes cálculos, cada servidor determina a melhor estratégia a seguir.

Neves [44] enfoca o PPRDR considerando de maneira conjunta a replicação, o posi-

cionamento das réplicas e distribuição eficiente das requisições. Em seu trabalho utiliza três abordagens distintas: o uso de métodos exatos para a versão *offline* através de formulações matemáticas e de um algoritmo de enumeração; o uso de metaheurísticas, também para a versão *offline* do problema; e métodos heurísticos e híbridos para a versão *online*, visando realizar um estudo mais aprofundado a respeito da melhor forma para tratar o problema real, que é tipicamente encontrado em ambientes de CDN. Seu trabalho trouxe as seguintes contribuições: 1-formulações matemáticas para várias versões do PPRDR, dentre as quais destaca-se a formulação FD, que contempla diversas características reais do problema, tais como: versão dinâmica do PPRDR, atendimento otimizado dos clientes, divisão de requisições em períodos, atendimento de requisições por múltiplos servidores, submissão e remoção de conteúdos; 2-Um algoritmo de Enumeração para resolução de instâncias de grande porte; 3-Duas versões da meta-heurística *Iterated Local Search* para a versão *offline* do problema. 4. Um *framework* unificador para heurísticas construtivas para a versão *online* do PPRDR tratado pela formulação FD; 5-Abordagens exatas e heurísticas para o PPR; 6-Abordagens exatas e heurísticas para o PDR; 7-Análise da complexidade do PDR, que prova que este problema pertence à classe P; 8-Abordagens híbridas compostas como combinação de abordagens para o PPR e para o PDR; e 9-Versões otimizadas (através da inclusão de componentes exatos) de heurísticas utilizadas em CDNs reais e apresentadas pela literatura.

Goldbach [29] propõe o Algoritmo POPMUSIC para resolver o PPRDR. POPMUSIC é uma meta-heurística proposta por Taillard et al. [50], um método geral projetado para lidar com instâncias grandes de problemas de otimização combinatória difíceis, através de sua decomposição em problemas menores. De acordo com Taillard et al. [50], não é fácil descobrir a maneira apropriada de decompor um problema, a priori. A decomposição de um problema em subproblemas pode levar a soluções de qualidade apenas moderada, já que os subproblemas podem ter sido criados de maneira arbitrária. POPMUSIC parte de uma solução disponível para o problema (não necessariamente boa), e tenta melhorar a qualidade da solução de partes da solução inicial, a posteriori.

Laoutaris et al. [37] apresentaram o PACA como possível solução para o problema de otimização de capacidade de armazenamento dentro do contexto das CDNs para topologias hierárquicas. A partir desta formulação, os autores formularam uma modelagem matemática sob forma de Problema de Programação Linear (PPL) para resolução do PACA. Em Laoutaris et al. [37] é utilizada uma aplicação do PACA para topologias hierárquicas ou em árvore, isto é, uma série de galhos inter conectadas onde as informações são buscadas em um servidor central e, caso não haja acesso direto, são realizados

redirecionamentos afim de buscar o servidor que o possua. O objetivo do modelo é desenvolver algoritmos que aproximam a performance ótima em vários cenários, porém que o faça de maneira mais eficiente possível. Para isso, são utilizadas diferentes técnicas: balanceamento de carga, a implementação do conceito de balanceamento de cargas entre os servidores de CDN, uma modificação proposta no modelo afim de que servidores de mesmo nível hierárquico poderiam encaminhar requisições uns para os outros. Através de resolução por um método heurístico, foi constatado redução de até 50% no custo total da solução quando os custos de comunicação entre os servidores de um mesmo nível hierárquico é suficientemente baixo.

Baseado na formulação proposta em Laoutaris et al.[37], Uderman et al. [53] propõem modificações no modelo para contemplar as características da topologia de rede sem hierarquia definida, onde todos os servidores estão aptos a interagir entre si e qualquer servidor pode ser designado para atender uma requisição, podendo ocorrer o atendimento simultâneo entre servidores. Uderman et al. [53], ao tratar do Problema da Alocação da Capacidade de Armazenamento em seu trabalho, destaca que este tipo de tratamento do espaço alocado nos servidores aumenta as chances de uma CDN ser capaz de realizar a entrega do conteúdo solicitado a partir de um servidor próximo ao cliente, visto que, conforme a demanda, esse servidor pode vir a ter maior espaço alocado. Além disso, reduz problemas de atrasos e diminui a carga de enlaces nas redes.

Boechat [17] e Boechat et al. [18] buscam reduzir o consumo de memória nas formulações de Neves [44], em que foram detectadas dois conjuntos de restrições redundantes na formulação matemática, denotada por Formulação Dinâmica (FD), proposta por Neves [44]. A retirada das restrições redundantes ocasionou uma redução de 6% no volume de equações do problema, mas ainda sim não foi possível resolver instâncias com mais de 50 servidores. Boechat [17] e Boechat et al. [18] descreve um novo problema dentro do contexto das CDNs, o Problema de Distribuição de Capacidades, Réplicas e Requisições, ou PDCRR. O PDCRR trata de maneira integrada a questão do posicionamento das réplicas nos servidores e a distribuição das requisições, mas paralelamente realiza a alocação dinâmica da capacidade de armazenamento dos servidores, as quais podem ser otimizadas para atender as variações de demanda nos diferentes períodos de tempo. Neste contexto, as características do PDCRR, como abordado Boechat [18], são: tratamento individual de requisições, requisições podem surgir ao longo do horizonte, capacidade de armazenamento pode ser alterada de acordo com a demanda e armazenamento dos servidores flexível ao longo do tempo. Em seu trabalho considera que as informações são conhecidas a priori, ou seja, as mudanças que ocorrem na rede, nos conteúdos e nas demandas são

conhecidas assim como também é conhecido quando estas mudanças ocorrerão o que faz com que o problema tratado no presente trabalho seja um problema offline.

Como pode ser observado pela Tabela 2.1, há poucos trabalhos que abordam mecanismos de decomposição no contexto das CDNs e além disso, apenas o presente trabalho, de acordo com o conhecimento dos autores, apresenta esta abordagem para o PDCRR.

Tabela 2.1: Trabalhos Relacionados

Ref	Problema	Modelo	Métodos		
			Exato	Heuristicos	Decomposição
Bektas et al. [14]	PDPS e PPRDR'	Estático	X	X	X
Tang et al. [51]	PPRDR	Estático		X	
Manoj et al.[49]	PPR	Dinâmico		X	
Aioffi et al. [11]	PPRDR'	Dinâmico	X	X	
Neves [44]	PPRDR	Dinâmico	X	X	X
Goldbach [29]	PPRDR	Dinâmico		X	X
Laoutaris et al. [37]	PACA	Estático	X	X	
Uderman et al. [53]	PACA	Estático		X	
Boechat [17, 19]	PDCRR	Dinâmico	X	X	
Este Trabalho	PDCRR	Dinâmico	X	X	X

Capítulo 3

LOCAL BRANCHING

3.1 *Local Branching* Clássico

O *local branching*, técnica proposta por Fischetti et al. [25], é um método de solução exato de PIMs que controla algum outro algoritmo de solução de PIMs exato, explorando de forma prioritária parcial ou completa da vizinhança de soluções viáveis já identificadas.

Este método tenta minimizar o tempo de resolução de grandes instâncias de problemas com tamanho real uma vez que tais problemas não são resolvidas otimamente em um tempo de computação aceitável. Nesse contexto, o *local branch* é a estratégia de solução que apresenta boas soluções viáveis ainda no início da computação, possibilitando que soluções que estejam a uma distância aceitável da solução ótima sejam encontradas utilizando um limite de tempo menor.

Um aspecto importante, mencionado pelos autores, é que o *local branching* melhora o comportamento heurístico do solver sem prejuízo à otimalidade do método, ou seja, utilizado em conjunto com o solver e dado o tempo de computação necessário, o *local branching* irá encontrar uma solução comprovadamente ótima.

3.1.1 Heurísticas para Fixação de Variáveis

Uma das técnicas utilizadas em heurísticas para solução de PIMs é denominada *hard variable fixing*. Utilizando um método de solução de PIMs (heurístico ou exato) uma solução inicial para o problema é encontrada rapidamente parametrizando o método para interromper a execução após pouco esforço computacional. A solução obtida pode ser até mesmo inviável para o PIM, como, por exemplo, a solução ótima da relaxação linear do problema. Observando os valores das variáveis na solução inicial parte das variáveis

é fixada em valor inteiro, por exemplo, fixando variáveis nas quais o valor inteiro mais próximo é não-nulo. O processo pode então ser repetido, observando que a cada iteração mais variáveis são fixadas em valores inteiros, diminuindo o tamanho do problema a cada iteração.

Um aspecto importante na técnica de *hard variable fixing* é a escolha de quantas e quais variáveis devem ser fixadas a cada iteração. Boas soluções normalmente são encontradas apenas após várias iterações fixando variáveis, sendo difícil determinar quais escolhas de fixação de variáveis das iterações anteriores foram ruins. Mesmo que escolhas de fixação ruins sejam percebidas, a criação de mecanismos para desfazer ou dirimir o impacto de escolhas de fixação ruins é uma tarefa complexa.

Como alternativa ao *hard variable fixing*, Fischetti et al. [25] propuseram uma técnica denominada *soft variable fixing*. Ao invés de escolher parte das variáveis e fixar seus valores, a técnica determina que uma proporção das variáveis de uma determinada solução viável deve ser fixada, delegando a tarefa de escolher as variáveis a serem fixadas para o solver. A motivação para realização do *soft variable fixing* é realizar a fixação de variáveis de forma tão efetiva quanto no *hard variable fixing*, porém permitindo um maior grau de liberdade para o solver com o objetivo de encontrar melhores soluções.

Supondo que exista uma solução viável para um PIM e que um percentual α das variáveis binárias não-nulas da solução devem ser fixadas, a restrição (3.1) pode ser adicionada ao problema para realizar *soft variable fixing*:

$$\sum_{j=1}^{n} \overline{x}_j x_j \geq \left\lceil \alpha \sum_{j=1}^{n} \overline{x}_j \right\rceil \tag{3.1}$$

Na restrição (3.1) o conjunto de variáveis binárias possui n variáveis $x_j : j \in \{1,\ 2,\ 3,\ \ldots,\ n\}$ e os valores das variáveis na solução viável é denotado por $\overline{x}_j : j \in \{1,\ 2,\ 3,\ \ldots,\ n\}$.

A restrição força que ao menos um percentual α das variáveis binárias não-nulas permaneçam com o mesmo valor nas novas soluções, o que é equivalente a dizer que apenas um percentual $(1 - \alpha)$ das variáveis binárias não-nulas na solução viável continuam livres para terem seu valor determinado pelo solver.

3.1.2 Heurística de Pesquisa de Vizinhança Variável

Uma das dificuldades apresentadas pelo método *local branching* é estabelecer e realizar buscas nas vizinhas afim de obter a solução ótima do problema. No trabalho Hansen et al. [33], os autores desenvolveram uma heurística de pesquisa de vizinhança variável (VNS- *Variable Neighborhood Search*) para resolver problemas de programação inteira mista usando o CPLEX como resolvedor. As vizinhança ao redor de uma solução inicial são definidas adicionando restrições ao problema original, como sugerido no *local branch*. Tanto o LB quanto o VNS usam as mesmas ferramentas: CPLEX e a mesma definição de vizinhança em torno do operador histórico. A diferença, segundo os autores, é que o VNS é mais simples e sistemático na exploração da vizinhança, obtendo melhorias com relação ao tempo de buscas em problemas difíceis.

O VNS, segundo Hansen et al. [31, 32], explora sistematicamente a ideia de mudança de vizinhança, tanto na descida aos mínimos locais quanto na fuga dos mesmos, utilizando a mesma definição de vizinhança em torno da solução incumbente, porém a principal diferença é que a mudança de vizinhança ocorre de forma mais sistemática, seguindo as regras do esquema geral de VNS.

Da análise feita por Hansen [33], utilizando uma notação matemática, no LB se o tempo total (t max) alocado para resolver uma determinada instância é alcançado antes que uma solução ótima seja encontrada, o LB se comporta como uma heurística, ou seja, a heurística para no tempo t max com a melhor solução conhecida como saída. Como a aplicação direta seria muito demorada para instâncias muito grandes, o espaço de solução é reduzido, dentro de um esquema de ramificação, pela introdução de novas restrições, especificando que no máximo k elementos podem ser permutados dentro da vizinhança. Depois de adicionar uma ou várias restrições lineares, o PIM resultante, que tem a mesma estrutura do PIM original, mas um espaço de solução menor, é resolvido pelo CPLEX, exatamente ou heuristicamente, dentro de um limite de tempo neste espaço de solução reduzido.

Detalhes do método

Para detalhar o funcionamento do *local branching* utiliza-se a notação apresentada em Fischetti et al. [25] e o exemplo de formulação genérica de PIM, (3.2)-(3.6), proposto em Bazaraa et al. [12]:

$$Min\ c^T \mathrm{x} \tag{3.2}$$

$$S.a.$$

$$Ax \geq b \tag{3.3}$$

$$x_j \in \{0,\ 1\}\ \forall\ j \in \beta \neq \phi \tag{3.4}$$

$$x_j \geq 0,\ inteiro\ \forall\ j \in \mathbb{G} \tag{3.5}$$

$$x_j \geq 0, \forall\ j \in \mathbb{C} \tag{3.6}$$

No modelo o conjunto de variáveis $N = \{x_1,\ x_2, \cdots, x_n\}$ foi particionado nos subconjuntos β, $\mathbb{G}$ e $\mathbb{C}$ que correspondem respectivamente ao conjunto de variáveis binárias, conjunto de variáveis inteiras e conjunto de variáveis contínuas. O subconjunto β deve ser não-vazio, quanto os subconjuntos $\mathbb{G}$ e $\mathbb{C}$ podem ser vazios.

Considerando uma solução viável de referência $\overline{x}$ para (3.2)-(3.6), o conjunto $\overline{S} = \{j \in \beta\ : \overline{x}_j = 1\}$ define o suporte binário de $\overline{x}$. Para qualquer parâmetro k inteiro positivo, a k-ésima vizinhança $N(\overline{x}, k)$ de $\overline{x}$ define um subproblema contendo o conjunto de soluções viáveis de (3.2)-(3.6) que satisfazem a restrição (3.7), chamada de restrição de *local branching:*

$$\delta\left(x, \overline{x}\right) : \sum_{j \in \overline{S}} \left(1 - x_j\right) + \sum_{j \in \beta \setminus \overline{S}} x_j \leq k \tag{3.7}$$

Na restrição (3.7) são contabilizadas as variáveis binárias que eram unitárias na solução de referência e se tornaram nulas, assim como as variáveis binárias que eram nulas e se tornaram unitárias. O número de "trocas" de valor nas variáveis binárias deve ser menor ou igual ao parâmetro inteiro k.

Para problemas onde a cardinalidade do suporte binário de qualquer solução viável de (3.2)-(3.6) é uma constante, a restrição (3.7) pode ser reescrita de forma simplificada conforme a restrição (3.8):

$$\delta\left(x, \overline{x}\right) : \sum_{j \in \overline{S}} \left(1 - x_j\right) \leq k' \tag{3.8}$$

Na restrição (3.8) são contabilizadas apenas as variáveis que eram unitárias na solução de referência e posteriormente se tornaram nulas. Em problemas em que a cardinalidade do suporte binário de qualquer solução viável é constante, sempre que é atribuído valor nulo a uma variável binária que antes era unitária é necessário também atribuir valor unitário a uma variável binária que antes era nulo. Devido a este comportamento a utilização da restrição (3.7) nestes problemas resultaria em uma contabilização de alterações em múltiplos de 2. Para estes problemas o efeito da utilização da restrição (3.7) ou (3.8) é o mesmo se $k' = k/2$.

A restrição de *local branching* pode ser utilizada como critério de *branching* em métodos enumerativos como o *branch-and-bound* e o *branch-and-cut*. Dada uma solução viável $\overline{x}$ o espaço de busca pode ser particionado utilizando a disjunção apresentada na Figura 3.1.

O tamanho inicial das vizinhanças, dado pelo parâmetro k, deve ser suficientemente grande para possibilitar que a vizinhança da solução viável contenha soluções melhores, porém precisa ser suficientemente pequeno para que seja possível explorar a vizinhança adequadamente considerando um pequeno limite de tempo.

O método de *local branching* é iniciado com a formulação original do problema e uma solução viável $\overline{x}_1$. Em seguida, uma nova restrição é adicionada, (ver restrição 3.7), centrada em $\overline{x}_1$ para reduzir o espaço de solução X, usando uma função de distância $d(x, \overline{x}_1)$, para o conjunto $X_1 = X \cap N_k(\overline{x}_1)$, o que corresponde ao ramo esquerdo da disjunção conforme a Figura 3.1.

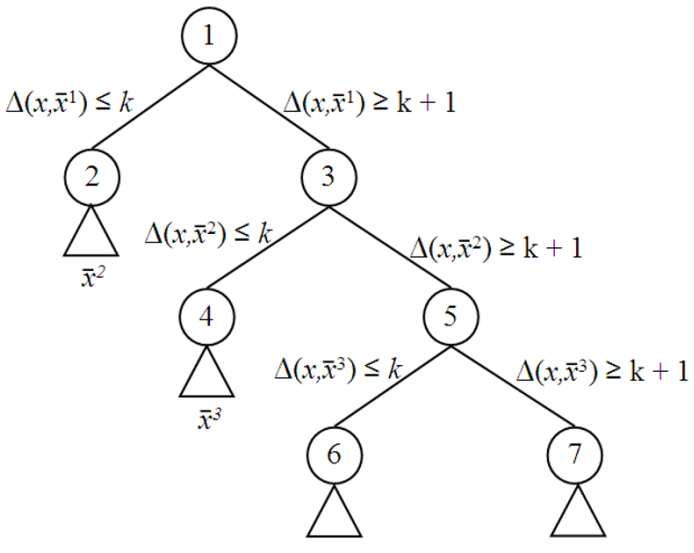

Figura 3.1: $\delta(x, \overline{x}) \leq k'$ $(ramoesquerdo)$ ou $\delta(x, \overline{x}) > k'$ $(ramodireito)$

Se uma solução melhor $\overline{x}_2$ for encontrada, um novo problema é gerado da seguinte maneira. A relação $\leq$ em (3.7) é substituída por $>$, e uma nova ramificação local a restrição (3.7) é adicionada, mas centrada em torno do novo titular $\overline{x}_2$ ($(x, \overline{x}_2) \leq k$). Assim, um conjunto viável X_2 para este subproblema é dado como:

$$X_2 = X \cap (N_k(\overline{x}_2) \setminus N_k(\overline{x}_1)) = (X \setminus X_1) \cap N_k(\overline{x}_2). \tag{3.9}$$

O resolvedor é chamado novamente, sucessivamente. Este procedimento de *ramificação* é iterado, desde que haja melhorias nos valores da função objetivo dentro do tempo do nó t: l novas restrições são adicionadas na iteração l e o conjunto viável fica da seguinte forma:

$$\mathrm{X}_l = (X \setminus X_1 \setminus X_2 \setminus \cdots \setminus X_l) \cap N_k(\overline{x}_l) \tag{3.10}$$

Nas Figuras 3.2 e 3.3 são ilustradas o comportamento do método, na qual o resolvedor é chamado para encontrar a solução ótima da vizinhança $\overline{x}_1$, $N(\overline{x}_1, k)$. Após a completa exploração da vizinhança a restrição de *local branching* (3.7) centrada em $\overline{x}_n$ é substituída pelo seu complemento, o que corresponde ao ramo direito da disjunção Figura (3.1), evitando que a vizinhança seja explorada novamente nas iterações seguintes.

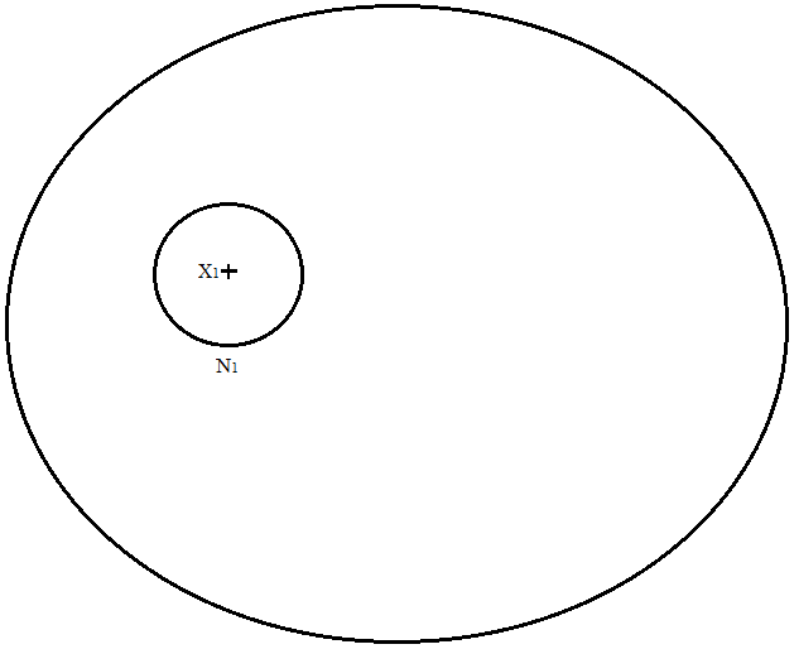

Figura 3.2: Vizinhaça N1 explorada após encontrar a solução inicial X1.

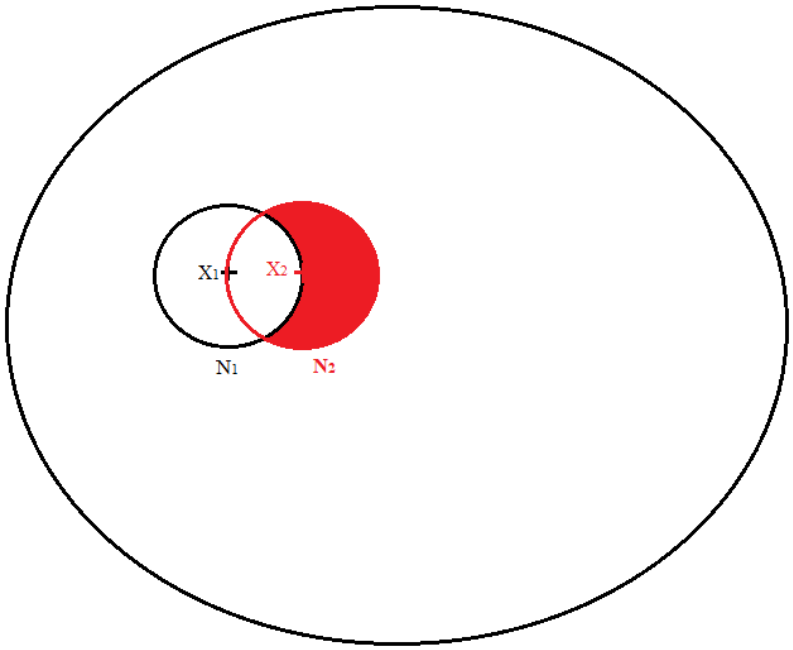

Figura 3.3: Vizinhaça N2 explorada após encontrar X2, restrita a vizinhaça N1.

No *local branching* clássico existem duas alternativas para o prosseguimento do método após a completa exploração de uma vizinhança qualquer $N(\overline{x}_n, k) = \{\mathrm{x} \mid d(x, \overline{x}_n) \leq k\}$:

- Caso o ótimo local $\overline{x}_{n+1}$ da vizinhança $N(\overline{x}_1, k)$ seja melhor do que $\overline{x}_n$:

  E adicionada então uma nova restrição de *local branching* (3.7) centrada em $\overline{x}_{n+1}$ e o solver é chamado para explorar completamente a nova vizinhança $N(\overline{x}_{n+1}, k)$ em uma nova iteração controlada pelo *local branching*.

- Caso o ótimo local $\overline{x}_{n+1}$ da vizinhança $N(\overline{x}_n, k)$ não seja melhor do que $\overline{x}_n$:

O resolvedor é chamado para explorar o restante do espaço de busca, deixando de ter comportamento controlado pelo *local branching*.

3.1.3 Extensões para o *Local Branching*:

O trabalho de Fischetti et al. [25] propõe diversas extensões ao método de *local branching* clássico como descrito abaixo:

- Limite de tempo e intensificação: um novo parâmetro $t_{\max}$ limita o tempo de exploração das vizinhanças. Caso o limite de tempo $t_{\max}$ seja atingido sem encontrar

a solução ótima da vizinhança $N(\overline{x}_n, k)$ podem acontecer dois casos: Caso a exploração tenha identificado uma solução viável $\overline{x}_{n+1}$ melhor que a solução atual $\overline{x}_n$, é criada uma nova vizinhança centrada em $\overline{x}_{n+1}$. E adicionada a restrição de *local branching* 3.7 centrada em $\overline{x}_{n+1}$ e removida a restrição centrada em $\overline{x}_n$ sem adicionar seu complemento. A adição do complemento da restrição centrada $\overline{x}_n$ iria remover a vizinhança $N(\overline{x}_n, k)$ do espaço de busca, o que seria matematicamente incorreto pois a vizinhança ainda pode conter uma solução ótima de boa qualidade. Caso a exploração não tenha identificado uma solução viável x_{n+1} melhor que a solução atual x_n o parâmetro k é reduzido, facilitando a determinação do ótimo local dentro do limite de tempo por realizar a exploração de uma vizinhança menor.

- Diversificação: pode ser utilizado quando a exploração de uma vizinhança $N(\overline{x}_n, k)$ não encontrou uma solução viável $\overline{x}_{n+1}$ melhor que a solução atual $\overline{x}_n$. Ao invés de chamar o solver para explorar o restante do espaço de busca, o parâmetro k é aumentado, possibilitando a identificação de uma solução viável melhor que a atual por realizar a exploração de uma vizinhança maior. Este tipo de diversificação é denominado *soft diversification*. Após o aumento do parâmetro k é possível que ainda assim não seja encontrada uma solução viável melhor que a atual. Nesse caso o *local branching* é reiniciado utilizando uma vizinhança centrada em uma solução pior do que a solução atual, permitindo a exploração de uma região diferente do espaço de buscas. Este tipo de diversificação é denominado *strong diversification*.

- Utilização de soluções não-viáveis: para problemas onde determinar uma solução viável inicial é difícil, é sugerida a criação de variáveis de folga para parte do conjunto de restrições, penalizando a inviabilidade na função objetivo.

- *Local branching* em variáveis inteiras não-binárias: para que seja possível realizar o *branching* em variáveis inteiras não-binárias é necessário criar variáveis de folga para representar a variação de cada variável inteira dentro de limites superior e inferior. A restrição de *local branching* precisa ser adaptada para contemplar esta alteração como detalhado em Fischetti et al. [25].

- *Utilização de solução inicial inviável*: em sua descrição original, é necessário uma solução viável para iniciar a computação utilizando o *local branching*. O método pode ser adaptado para tratar soluções iniciais inviáveis.

- Uma outra extensão proposta para o *local branching* é o *Relaxation induced neighbourhood search* (RINS). O RINS realiza a fixação de variáveis comparando os valores

atribuídos ás variáveis binárias em determinada solução viável e os valores que estas variáveis assumem na relaxação linear do subproblema atual. RINS explora a idéia de que fixar o valor de variáveis que possuem o mesmo valor na solução viável e na relaxação do subproblema atual provavelmente leva a boas soluções, concentrando o esforço computacional do resolvedor em variáveis que diferem na solução viável e na relaxação linear, sendo as variáveis que permaneceram livres as que intuitivamente podem causar uma melhoria mais significativa na função objetivo.

3.2 Implementação do Algoritmo *Local Branching*

Para a implementação do método foram criadas, além da função principal que carrega os dados do problema e cria as estruturas de variáveis, vetores e matrizes, as seguintes funções: **solver_i, sup_binario, local_branch, complem_lb e solver_f.**

Cada uma das funções implementadas realiza as seguintes ações:

a) **solver_i** - configura o resolvedor para encontrar uma primeira solução do problema, a qual servirá como ponto de partida para definir os conjuntos β, $\mathbb{G}$ e $\mathbb{C}$, para iniciar o método, encontrar um suporte binário e definir a primeira vizinhança.

b) **sup_binario** - captura os índices das variáveis de uma determinada solução cujos valores assumiram o valor um, definindo desta forma o suporte binário e consequentemente permitindo a geração de uma vizinhaça.

c) **local_branch** - utiliza a informação dos índices capturados pela função sup_binario e acrescenta uma nova linha de restrição ao problema, definindo uma vizinhança de busca e utiliza o resolvedor para encontrar a solução dentro da vizinhança respeitando as restrições do problema.

d) **complem_lb** - limita o espaço de busca excluindo a vizinhança já explorada pela função local_branch, criando um histórico, o qual armazena as informações destas vizinhanças, evitando que buscas em espaços já explorados sejam realizadas, tornando o processo mais eficiente.

e) **solver_f** - é chamada quando não se consegue melhorias nas soluções utilizando a técnica do *local branch*. Neste caso, o resolvedor explora o restante do espaço de busca que não foi analisado nas vizinhanças dos suportes binários.

Conforme descrito no **Algoritmo 1**, o código inicia configurando o resolvedor e carregando os dados de inicialização do problema, ou seja, todas as variáveis e restrições. Após está primeira etapa, é executada a função **solver_i**, que, dado um limite temporal, encontra a Solução 1. Para esta solução, é verificada sua otimalidade, caso seja a Solução 1 a solução ótima do problema, o código é interrompido e a solução encontrada.

Se a Solução 1 não for a solução ótima, então a Solução 1 é armazenada na Solução 2, a qual será analisada pela função **sup_binario** para identificar o conjunto das variáveis binárias que assumiram o valor 1, formando assim o suporte binário e gerando uma vizinhança para que o resolvedor procure uma solução ótima, Solução 3, nesta vizinhança, através da função **local_branch**. Caso a Solução 3 seja melhor que a Solução 2, o código executa a função **complem_lb** para excluir a vizinhança que foi explorada para encontrar a Solução 3, evitando que o código fique preso a uma mesma solução. Então a Solução 3 é armazenada na Solução 2 e o processo se repete.

Caso a Solução 3 seja pior que a Solução 2, o código atinge o limite do contador dentro *looping*, encerrando-o, e então executa a função **solver_f**, a qual busca a solução ótima no restante do espaço de soluções viáveis, e dentro do limite do tempo de 03 horas, encontrando a Solução 4. Então é feita uma comparação entre as soluções e o código informa se a mesma é a solução ótima ou a solução encontrada dentro do limite de tempo estabelecido.

Algoritmo 1 Função LocalBranching(tempo inicial, tempo tolerável, k)

```
1: Inicialização;
2: Solução 1 = solver_i ( tempo inicial )
3: se Solução 1 for ótima então
4:    retorne Solução 1
5: senão
6:    Solução 2 = Solução 1
7:    contador = 1
8:    enquanto (Tempo de execução < tempo tolerável e contador < 2) faça
9:       sb = sup_binario (Solução 2)
10:      Solução 3 = local_branch (sb, Solução 2, k)
11:      se Solução 3 for pior que solução 2 então
12:         contador = contador +1
13:      senão
14:         complem_lb (Solução 3, histórico)
15:         Solução 2 = Solução 3
16:      fim se
17:   fim enquanto
18:   Solução 4 = solver_f (histórico)
19:   se solução 4 for pior que Solução 3 então
20:      retorne Solução 3
21:   senão
22:      retorne Solução 4
23:   fim se
24: fim se
```

Capítulo 4

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS

4.1 Algoritmo Genético de Chaves Aleatórias Viciadas

Com o intuito de obter melhores resultados nas instâncias maiores do problema proposto neste trabalho, optou-se por utilizar uma meta-heurística para atingir tal objetivo. Na sua definição original, segundo Gendreau et al.[28], métodos meta-heurísticos são métodos que coordenam outros métodos para a busca de soluções ótimas locais e com estratégias de alto nível para fugir desses ótimos locais, dessa forma, possibilita ter uma maior chance de se encontrar uma solução ótima global ou mais próxima dela, sem haver a necessidade de explorar todo o espaço de soluções de um problema. Alguns exemplos de meta-heurísticas mais utilizadas são *Greedy Randomized Adaptive Search Procedure* (GRASP), Busca Tabu, Busca Local Iterada e Busca de Vizinhança Variável.

A meta-heurística aplicada para o PDCRR, neste trabalho, foi o Algoritmo Genético de Chaves Aleatórias Viciadas ou *Biased Random-Key Genetic Algorithm*, BRKGA, recentemente proposto por Gonçalves et al.[30], sendo uma evolução do Algoritmo Genético de Chaves Aleatórias ou *Random-Key Genetic Algorithm*, RKGA, proposto por Bean et al. [13].

O algoritmo genético é tido como um algoritmo evolutivo, categorizando uma classe de meta-heurística inspirada no conceito da Teoria da Seleção Natural proposta por Darwin [21]. Segundo Eiben et al. [23], nesses algoritmos existem uma série de indivíduos pertencentes a uma população que, devido a uma pressão causada pelo ambiente, somente os mais adaptados participarão das próximas gerações desta população, gerando uma melhor

adaptação da população em geral ao ambiente específico. Esses algoritmos, geralmente, são organizados de tal maneira que cada indivíduo da população é uma possível solução, e a pressão que o ambiente causaria para estimular a escolha dos mais aptos, usualmente, é uma função objetivo que mede a qualidade dos indivíduos, indicando se este é mais apto ou não para maximizar ou minimizar o valor da solução de um problema. São utilizados também mecanismos para criar diversidade nas características dos indivíduos, como, por exemplo, mutação das características de um indivíduo e seleção de alguns indivíduos que não são os melhores de uma geração para reprodução. Com isso, facilita-se escapar de soluções ótimas locais e gera-se maior chance de se aproximar de soluções ótimas globais.

A proposta presente nos algoritmos genéticos consiste em estabelecer uma população inicial de "cromossomos" ou indivíduos, que são decodificados em vetores de valores, os "genes". A cada geração, um grupo dos cromossomos é escolhido para se reproduzir e frequentemente os mais aptos, os de melhor valor da função objetivo, produzem mais descendentes do que os demais. Outro procedimento existente no algoritmo genético é a recombinação, *crossover*, dos genes desses cromossomos, na qual é feita imitando o conceito biológico, copiando subpartes, genes, de cada um dos cromossomos envolvidos na geração de um descendente. Operadores de mutação também são utilizados para gerar diversidade dentre as novas gerações, aleatoriamente mudando valores dos genes pertencentes a um cromossomo.

Tanto o RKGA quanto o BRKGA, segundo Gonçalves et al. [30], utilizam um sistema de chaves aleatórias para codificar as soluções (indivíduos ou cromossomos), em que cada cromossomo é composto por uma sequência de chaves aleatórias, tipicamente no intervalo [0, 1). Essas soluções com chaves aleatórias podem ser decodificadas para o espaço de soluções do problema específico a ser resolvido, possibilitando que os dois algoritmos operem diretamente sobre as chaves e não sobre soluções específicas do problema, criando maior desacoplamento do algoritmo em relação ao problema específico a ser resolvido. Este método também evita o problema de geração de descendentes inviáveis, já que ele representa codificações dos cromossomos de um modo indireto. Além do uso de chaves aleatórias, estes algoritmos também utilizam o conceito de elitismo para a seleção de quais cromossomos são os "melhores" de uma geração, onde parte dos cromossomos da geração atual será copiada para a próxima geração, assegurando que as melhores soluções sejam levadas adiante nas novas gerações. No BRKGA, a escolha dos cromossomos para a recombinação é feita de forma aleatória, assim como no RKGA, entretanto, exige-se que um dos escolhidos seja pertencente à elite das soluções, já o outro cromossomo escolhido pode ser de qualquer parte da população. Outro detalhe é que na fase de recombinação

do par de indivíduos, o gene provindo do indivíduo da elite deve ter sempre maior chance de ser escolhido sobre o gene do segundo indivíduo.

Comparações feitas por Silva et al. [46] demonstraram que, no geral, o BRKGA converge mais rapidamente em direção à solução ótima desejada do que o RKGA, indicando ser uma meta-heurística mais eficiente. O BRKGA pode ser empregado em problemas que precisem de soluções rápidas ou em tempo real e, segundo Silva et al. [46], as aplicações de sucesso do BRKGA são reportadas em diversas áreas, incluindo Telecomunicações, Transportes, Leilões combinatórios, Problemas de empacotamento, Escalonamento de tarefas e Engenharia de produção em geral.

Características específicas do BRKGA são a representação de seus indivíduos como vetores de chaves aleatórias e maior probabilidade de passar as características genéticas dos indivíduos pertencentes à elite para as novas gerações. Este método, estabelecido por Silva et al. [46], é composto por cinco etapas as quais serão descritas a seguir:

1. Geração de chaves aleatórias: inicialização da população, etapa de criação dos vetores de chaves aleatórias;

2. Decodificação de vetores de chaves aleatórias: etapa que codifica os vetores do espaço de chaves aleatórias para o espaço de soluções reais do problema;

3. Classificação da população em elite e não elite: após a geração de uma nova população, esta é ordenada e dividida em dois conjuntos de indivíduos (elite e não elite), de acordo com a aptidão das soluções. A elite é então selecionada para ser copiada para a nova geração sem alterações;

4. Mutação ou geração de mutantes: uma parte da nova geração é formada por indivíduos gerados aleatoriamente. Nesta etapa alguns mutantes são inseridos na nova geração;

5. Cruzamento ou recombinação de indivíduos: o restante dos indivíduos é gerado selecionando aleatoriamente um indivíduo da elite e outro não pertencente à elite, para recombinar os genes dos dois indivíduos, porém, sempre tendendo a escolher um gene do indivíduo pertencente à elite.

A primeira etapa é executada apenas uma vez na inicialização da técnica. Já as etapas de 2 a 5 são executadas repetidamente até que uma determinada condição de parada seja satisfeita, tal como número máximo de gerações ou convergência da solução para um valor

limite. O diagrama apresentado pela Figura 4.1 ilustra o esquema de funcionamento do BRKGA, apresentando o fluxo de execução do BRKGA, adaptado de Gonçalves et al. [30].

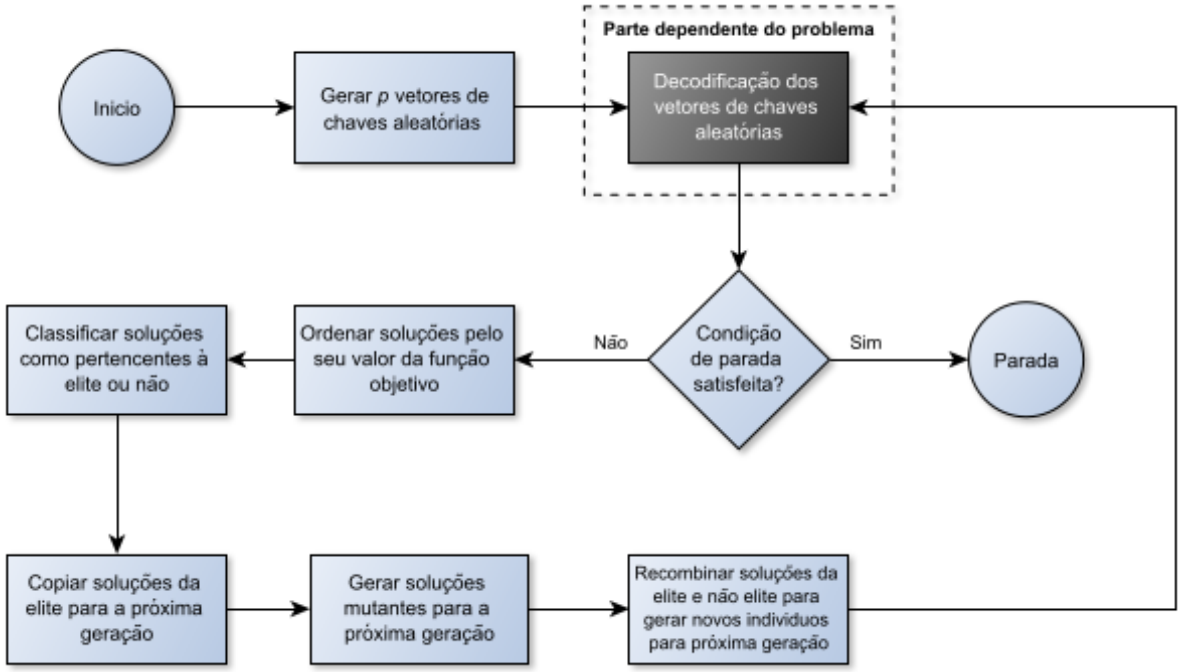

Figura 4.1: Esquema de funcionamento do BRKGA, Gonçalves et al. [30].

Na etapa de geração de chaves aleatórias é inicializada a população a ser evoluída pela meta-heurística. Esta população é formada por p vetores de n chaves aleatórias, tal que cada vetor representa um indivíduo ou cromossomo. Cada um dos n genes pertencente ao cromossomo é gerado aleatoriamente por uma chave que pode assumir valores no intervalo de $[0, 1) \in \mathrm{R}$. Na Figura 4.2, pode-se ver um exemplo de um cromossomo representado pelo vetor X.

0,65	0,24	0,79	0,00	0,91	0,47
X_1	X_2	X_3	X_4	X_5	X_6

Figura 4.2: Exemplo de cromossomo com seis chaves aleatórias, Gonçalves et al. [30].

A codificação do cromossomo como um vetor de chaves aleatórias é importante, pois cria desacoplamento da meta-heurística em relação ao problema, tal que o algoritmo pode lidar com chaves aleatórias diretamente e somente uma pequena porção do algoritmo trata a parte especifica do problema, aumentando o reaproveitamento total de vários

componentes da técnica. Para cada nova aplicação é necessário somente criar a parte referente à decodificação, vide Figura 4.1.

Após a inicialização da população, a decodificação dos cromossomos é executada. Esta etapa é a parte com maior dependência do problema a ser tratado, devendo mapear o cromossomo do espaço de solução de chaves aleatórias para o espaço de solução do problema. Como pode ser observado na Figura 4.3, o papel do decodificador é mapear cromossomo (como o vetor X da Figura 4.2) de chaves aleatórias para o espaço de soluções específico ao problema e deve ser implementado de modo distinto em cada problema. Somente depois da fase de decodificação é possível calcular a aptidão de cada cromossomo.

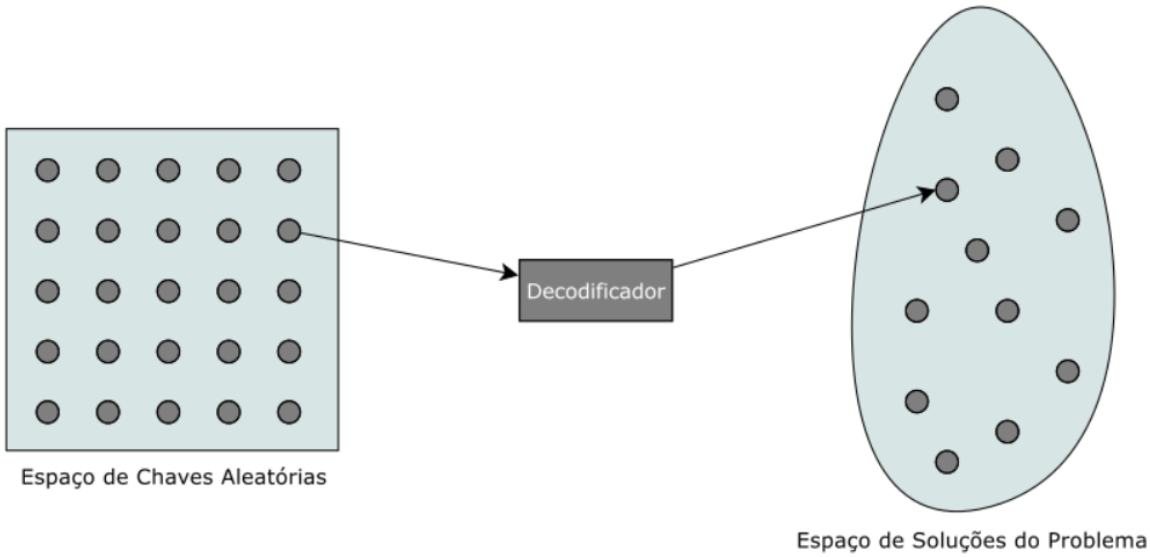

Figura 4.3: Esquema de funcionamento do decodificador, Gonçalves et al.[30].

A Figura 4.3 mostra o esquema de funcionamento do decodificador, que mapeia as soluções do espaço de chaves aleatórias para o espaço de soluções do problema.

Para melhor ilustrar o funcionamento do BRKGA será utilizado como exemplo prático o problema da mochila binária onde a decodificação pode ser modelada representando os itens selecionados para adição à mochila seguindo a ilustração da Figura 4.2, de forma que X representa o vetor de chaves aleatórias e X_i a i-ésima chave aleatória a ser decodificada, traduzindo como 0 se o item não for selecionado, (4.1), e 1 caso contrário, (4.2), segundo os critérios:

$$X_i < 0,5 \rightarrow 0 \tag{4.1}$$

$$X_i \geq 0,5 \rightarrow 1 \tag{4.2}$$

A Figura 4.4 ilustra como o cromossomo, exemplificado pela Figura 4.2, fica após a decodificação.

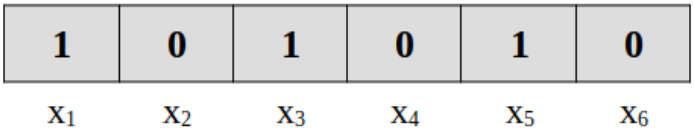

Figura 4.4: Exemplo de um vetor X, Fig. 4.2, decodificado, Gonçalves et al. [30].

Depois dos cromossomos terem seus respectivos valores de aptidão calculados na etapa de decodificação, eles são ordenados e posteriormente classificados em uma parcela que pertencente à elite, que comporta os p melhores cromossomos e o restante não pertencente a elite. Essa parcela da população com as melhores soluções é copiada intacta para a próxima geração. Com isso, o conceito de elitismo força parcialmente que a meta-heurística busque por soluções similares às que estão demonstrando serem as melhores até então. A recomendação feita por Gonçalves et al. [30] é que a proporção de cromossomos elite na população seja tal que $0,10p \leq pe \leq 0,25p$ para se obter bons resultados.

Na Figura 4.5, que refere-se a um problema de maximização, é ilustrada a classificação da população da geração atual, em cromossomos que fazem parte da elite e os que não fazem.

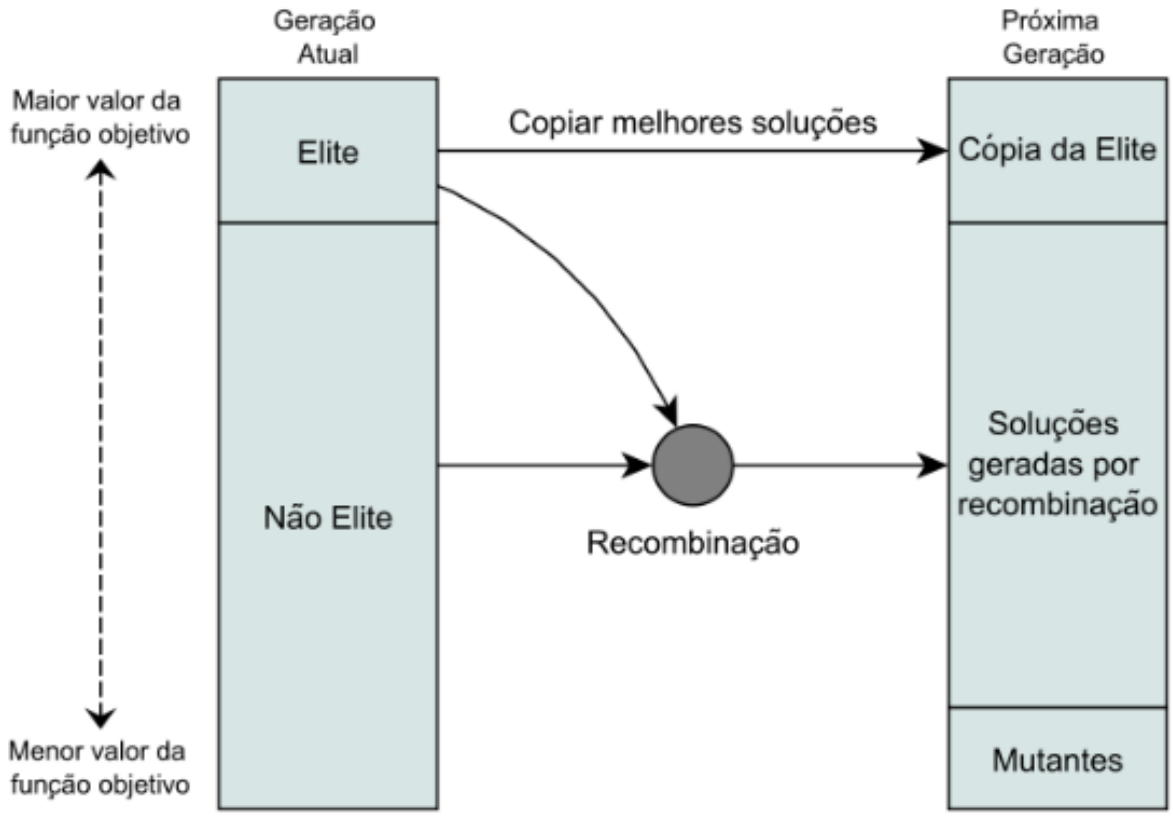

Figura 4.5: Classificação da população, Gonçalves et al. [30].

Uma pequena parte da nova geração é produzida aleatoriamente para introduzir diversidade ao processo evolutivo e evitar a convergência prematura para ótimos locais, adicionando cromossomos mutantes na próxima população. Na Figura 4.5, é ilustrada a porção da próxima geração composta por estes cromossomos mutantes. Segundo Gonçalves et al. [30], recomenda-se que a proporção de mutantes na próxima geração seja tal que $0,10pop \leq pop_n \leq 0,30pop$ para que sejam obtidos bons resultados.

Os indivíduos restantes da próxima geração são produzidos na etapa de recombinação. No caso do BRKGA, é utilizado o método *Parametrized Uniform Crossover*, Spears et al. [48], que consiste em sortear de qual cromossomo pai deve ser herdado cada gene. Para que a meta-heurística tenha a característica *biased*, ou viciada, um dos cromossomos pai deve ser escolhido aleatoriamente, entretanto, sempre pertencendo à elite e o outro deve ter sua escolha aleatória, porém, advindo do grupo dos que não pertencem a elite. Além disso, no processo de recombinação dos genes, a probabilidade γe de um gene ser herdado do cromossomo da elite deve ser sempre maior ou igual do que a chance de um gene ser herdado do cromossomo não pertencente à elite, ou seja $\gamma e \geq 50\%$.

Na Figura 4.6 é ilustrado o processo de recombinação no BRKGA, tal que dois cromossomos são escolhidos para serem recombinados, seguindo as regras anteriormente citadas e, para cada gene, um número aleatório é sorteado entre 0% e 100%. Como tem-se $\gamma e = 70\%$, se o número for menor ou igual a 70%, então, é herdado o gene do cromossomo da elite, se for maior que 70%, então, o gene é herdado do outro cromossomo.

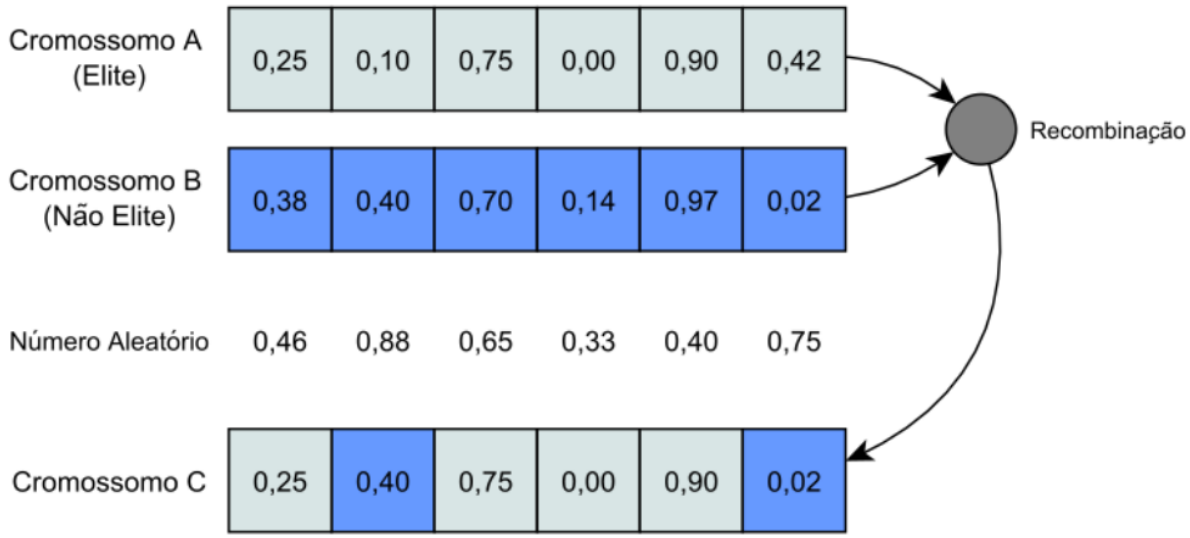

Figura 4.6: Exemplo de recombinação, com $\gamma e = 70\%$, Gonçalves et al. [30].

O valores recomendados em Gonçalves et al. [30] para a probabilidade de um gene

ser escolhido do progenitor pertencente à elite é de $50\% \leq \gamma e \leq 80\%$.

Para que o algoritmo não execute indefinidamente, é necessário determinar qual é a condição que deve ser satisfeita para que a execução termine. Existem vários tipos de condição de parada, dentre elas:

- Número fixo de gerações desde o início da execução do algoritmo;
- Número fixo de gerações desde que houve a última melhora na qualidade da melhor solução;
- Tempo máximo de execução;
- Encontrar uma solução com aptidão melhor ou igual a um valor estabelecido.

Esta parte do algoritmo, apesar de independente do problema, deve ser adaptada de acordo com as necessidades intrínsecas da aplicação, visto que alguns critérios são mais adequados a algumas situações do que outras. Como a escolha da condição de parada geralmente envolve em determinar parâmetros que façam o algoritmo dar a melhor solução dentro de um prazo de tempo ou número máximo de gerações, pode-se utilizar também de paralelização de alguns componentes do algoritmo para acelerar o tempo de execução ou aumentar a chance de se encontrar mais soluções de qualidade.

4.2 Estratégia para utilização do BRKGA no PDCRR

A estratégia para aplicar o BRKGA no PDCRR na fase de decodificação dos indivíduos e obtenção do valor da sua aptidão determinado pela função objetivo, será similar a apresenta por Neves et al. [43] para resolver o PPRDR. Em Neves et al. [43], o problema é dividido em duas partes: primeiro, uma formulação matemática é usada para determinar o posicionamento das réplicas e, na sequência, de posse do posicionamento das réplicas informado pela formulação, um algoritmo de fluxo em rede é usado para resolver a distribuição das requisições. Neste trabalho, o algoritmo genético é usado para determinar o posicionamento das réplicas, que, indiretamente, resolve o problema de alocação das capacidades, pois uma vez posicionadas as réplicas com as capacidades disponíveis, encontra-se uma solução viável para a alocação das capacidades, em seguida, a distribuição das requisições é feita via fluxo em rede. Caso a soma dos espaços alocados pelas réplicas ultrapasse o espaço disponível, a solução é penalizada conforme descrito na Seção 4.2.1..

O método exato para resolver o PDR que consiste em resolver um Problema de Fluxo de Custo Mínimo (PCFM) em uma rede específica, Ahuja et al. [10]. Nessa rede, os vértices representam solicitações, servidores, fonte de fluxo e coletor de fluxo. A demanda está associada a cada vértice dependendo do que ele representa. Cada arco na rede tem um custo, uma capacidade e uma classe. O custo é o valor pago por cada unidade de fluxo que atravessa o arco e a capacidade é a quantidade máxima de fluxo permitida no arco. Como os vértices representam diferentes entidades do problema, os relacionamentos entre eles nem sempre são iguais. Uma solução do PCFM nesta rede é equivalente a uma solução do PDR fornecido pela formulação matemática exposta em Neves [42].

A Figura 4.7 mostra a rede construída seguindo as etapas mencionadas para uma instância com duas solicitações de clientes, vértices C1 e C2, e dois servidores, vértices S1 e S2.

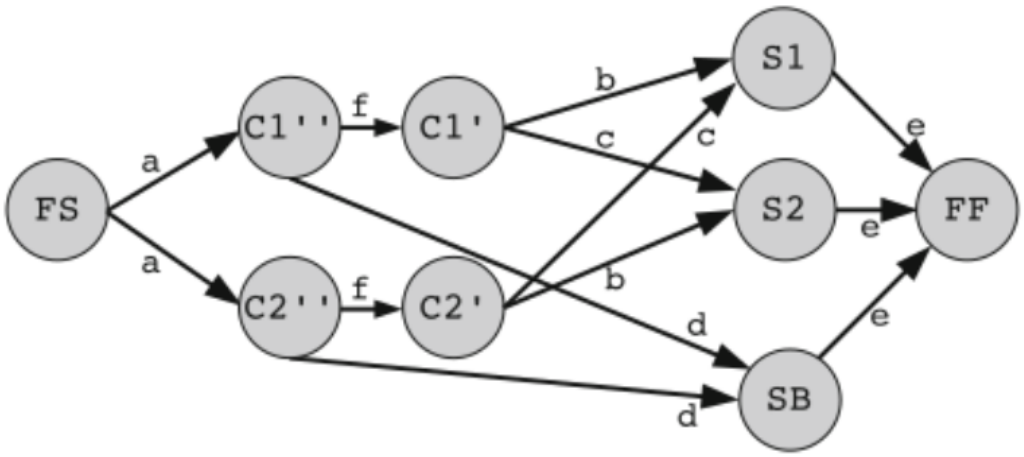

Figura 4.7: Modelo de fluxo em rede, Neves et al. [43].

Neste exemplo contido em Neves et al. [43], S1 tem a réplica desejada por C1 e S2 tem a réplica desejada por C2, os vértices representam requisições, servidores, origem de fluxo, (FS), e destino de fluxo, (FF). Demandas específicas são associadas aos vértices de acordo com o que eles representam. Os arcos que ligam os vértices possuem um custo associado, uma capacidade e uma classe. O custo é o valor pago para transmitir uma unidade de fluxo pelo arco e a capacidade representa a quantidade máxima de fluxo que o arco pode transportar. Já as classes dos arcos representam as relações entre os vértices, sendo os arcos de origem que ligam o vértice (FS) aos clientes pertencem à classe a e modelam a demanda dos clientes, os arcos entre os vértices que representam os clientes são da classe f e modelam a capacidade de banda do cliente, os arcos que ligam os clientes aos servidores são das classes b, c e d e modelam, respectivamente, o atendimento viável, o atendimento

inviável e o *backlog* da requisição, e os arcos da classe *e* modelam as capacidades dos servidores. Como os vértices representam entidades diferentes do problema a relação entre eles nem sempre é a mesma, e esta diferença entre os relacionamentos é modelada pelas classes de arcos. Maiores detalhes sobre a construção da rede podem ser vistos em Neves et al. [43].

4.2.1 Penalidades

Quando trabalhamos com métodos heurísticos podemos acabar nos deparando com soluções que estão fora do espaço das soluções viáveis, ou seja, soluções que não respeitam as restrições do problema. Para contornar estas situações, são criadas penalizações para soluções que se apresentam nestas condições, fazendo com que não produzam bons resultados e sejam descartadas naturalmente pelo método, que no caso do BRKGA, não façam parte do grupo de elite e passem a ser considerados como maus indivíduos para a espécie.

Para o PDCRR há duas situações que criam soluções com tais características: 1) quando a soma dos espaços que é consumido pelos servidores for maior que o limite de espaço disponível; e 2) quando um conteúdo é entregue fora do seu tempo de vida a um cliente (devido aos atrasos no atendimento), ou seja, os dados são removidos da CDN antes de serem entregues na totalidade ao cliente destino. No modelo matemático tratado neste trabalho há duas restrições referentes a estes dois casos, são elas (2.8) e (2.18), conforme descritas:

$$\sum_{j \in S} L_{G(i)} x_{ijt} - b_{i(t-1)} + b_{it} = D_{it} \ , \ \forall i \in R, \ \forall t \in \left[B_{G(i)}, E_{G(i)}\right],$$

$$\sum_{k \in C} L_k y_{kjt} \leq r_{jt} \ , \ \forall j \in S, \ \forall t \in T.$$

As restrições (2.8) e (2.18), respectivamente, garantem que a demanda será totalmente atendida, fazendo um controle que indica que a quantidade entregue do período atual somada à penalidade do período posterior deverá ser equivalente à demanda do período atual somada às penalidades por atraso do período anterior; e que a soma do espaço ocupado em um servidor não deve ultrapassar o espaço alocado neste servidor.

Então, a título de penalidade, quando estas restrições são violadas, é acrescentado à função objetivo a soma dos conteúdos perdidos multiplicado por 10000 e a soma das

quantidades de espaços em disco violadas em cada servidor multiplicado por 100 vezes o maior valor do custo em realizar *backlog*.

4.2.2 Melhoria no Desempenho do BRKGA

Para melhorar o desempenho do BRKGA foi usada a técnica da inserção de um indivíduo de boa qualidade na população inicial, Rochman et al.[45]. Esse procedimento foi realizado substituindo o pior individuo da população inicial, gerada aleatoriamente, por um indivíduo gerado pela heurística HNH adaptada ao PDCRR, Neves et al. [43].

Apesar de ter sido utilizada a heurística HNH, Neves et al. [43], a mesma não foi projetada para lidar com o PDCRR, e sim com o PPRDR. Embora os problemas sejam similares, as diferenças entre eles fazem com que o desempenho da heurística HNH não seja tão bom quanto o obtido no problema para o qual ela foi proposto.

Para obter melhor desempenho, é proposta neste trabalho uma adaptação da heurística HNH para o PDCRR. A heurística HNH é dividida em duas partes:

1. Uma formulação matemática que define o posicionamento das réplicas e a alocação das capacidades para um período; e
2. Algoritmo de fluxo em rede que distribui as requisições pelos servidores.

A adaptação proposta está no modelo matemático que define o posicionamento das réplicas. Como ambos os problemas (PDCRR e PPRDR) lidam da mesma forma com a distribuição de requisições, o algoritmo de fluxo em rede pôde ser reaproveitado de maneira integral.

A formulação matemática que posiciona as réplicas na heurística HNH original é a seguinte, considerando:

- m o número de servidores.
- n o número de conteúdos.
- p_{jk} a demanda do servidor j pelo conteúdo k.
- w_k o tamanho do conteúdo k.
- c_j o espaço em disco do servidor j.

$$Max \quad \sum_{j=1}^{m}\sum_{k=1}^{n} p_{jk}x_{jk} \tag{4.3}$$

$S.a.$

$$\sum_{k=1}^{n} w_k x_{jk} \leq c_j, \quad j \in M = \{1, ..., m\}, \tag{4.4}$$

$$\sum_{j=1}^{m} x_{jk} \geq 1, \quad k \in N = \{1, ..., n\}, \tag{4.5}$$

$$x_{jk} \in \{0, 1\}, \quad \forall j \in M, \forall k \in N. \tag{4.6}$$

Esta formulação não considera o custo do espaço necessário para posicionar as réplicas nos servidores, custo este presente na definição do PDCRR. Neste sentido a função objetivo da formulação matemática foi adaptada, como pode ser visto em (4.7):

$$Max \sum_{j=1}^{m}\sum_{k=1}^{n}(p_{jk} - \lambda \times w_k)x_{jk} \tag{4.7}$$

Onde λ é um peso arbitrário usado para contabilizar a importância dos custos de alocação na função objetivo do posicionamento de réplicas. Note que qualquer valor positivo para λ fará com que a formulação matemática avalie a relação custo benefício entre o tamanho ocupado por um conteúdo e a demanda que ele poderá suprir localmente em um determinado servidor.

Assim, a heurística HNH adaptada foi executada com múltiplos valores do parâmetro λ (no intervalo de [0, 0.9] com passo 0.1), encontrando assim, múltiplas soluções para o problema. A solução com melhor função objetivo é inserida na população do algoritmo genético.

Capítulo 5

RESULTADOS COMPUTACIONAIS

5.1 Equipamento Utilizado para Realizar os Testes nas Instâncias

Para realizar os testes das implementações do algoritmo *local branching* e do algoritmo genético de chaves aleatórias viciadas foi utilizado um *Notebook*, i7 de sexta geração, com 4 núcleos, 2.5 GHz de processamento por núcleo, memória 16 GB do tipo DDR4 e com sistema operacional Linux, versão 20 de 64 bits. As implementações foram realizadas em linguagem C++, utilizando o compilador G++ versão 4.3, o pacote de otimização CPLEX, versão 12. Para facilitar a utilização do BRKGA, foi desenvolvida por Toso et al. [52] uma *Application Programming Interface* (API), ou biblioteca, para a linguagem C++, que implementa a maior parte das rotinas necessárias para a utilização do BRKGA.

Como forma de permitir a comparação entre os resultados alcançados pelos Modelo Exato, o *Local Branch* e a heurística BRKGA, foi definido um indicador chamado *IND* que representa a distância percentual comparando as respostas obtidas pelos modelos implementados, sendo que quanto menor este valor, em módulo, mais próximo da solução encontrada pelo modelo exato o método chegou e quando seu valor é negativo significa que houve uma melhora em relação a solução apresentada pelo modelo exato. A sua fórmula pode ser vista em (5.1), sendo *FO* a Função Objetivo para cada instância do problema.

$$IND(\%) = \frac{\textit{Valor FO do (LB ou BRKGA) - Valor FO do modelo exato}}{\textit{Valor FO do modelo exato}} * 100 \qquad (5.1)$$

5.2 Instâncias de Testes

Foram utilizadas as instâncias disponíveis pelo Laboratório de Inteligência Computacional da UFF [1], as quais foram utilizadas em vários trabalhos, Neves [44] e Boechat [18], para testar o modelo proposto pela formulação, estas instâncias possuem números variados de servidores, conteúdos e requisições. Este conjunto de instâncias é o primeiro conjunto que considera simultaneamente várias características próximas à realidade, como, requisitos de QoS, capacidade de servidores diferentes e conteúdos dinâmicos.

Conforme citado em Neves [44], estas instâncias possuem índices que variam em uma escala de 01 a 20, divididas em quatro Classes, A, B, C e D, as quais serão expostas a seguir:

Classe A - são instâncias em escala reduzida, utilizadas para testes e por isso praticamente todos os valores utilizados para esta classe são escolhidos de maneira arbitrária. Os índices desta Classe variam de 01 a 05.

Classe B - são instâncias construídas com base em valores encontrados na literatura e com base em equipamentos de mercado disponíveis quando estas instâncias foram criadas (final de 2008). Os índices desta Classe variam de 06 a 10.

Classe C - são instâncias similares às instâncias da classe B, contudo, nas instâncias da classe C os servidores possuem restrições mais rígidas na capacidade de armazenamento. Os índices desta Classe variam de 11 a 15.

Classe D - são instâncias com restrições mais severas em termos de capacidade de armazenamento e em termos de banda nos servidores. Os índices desta Classe variam de 16 a 20.

Para cada possível tamanho de 10, 20, 30 ou 50 servidores, utilizadas para implementação do modelo do PPRDR, foram utilizadas as quatro classes. Após a realização de vários testes computacionais, foi constatado por Neves [44] que as instâncias das Classes A e B são de fácil resolução. Para todas as instâncias, considerou-se que o tamanho dos períodos é de 60 segundos.

Características das instâncias geradas por Neves [44] e citadas em seu trabalho para cada classe:

1. Períodos de tempos – foram considerados 15 períodos de tempos para as instâncias de classe A e 35 para as classes B, C e D.

2. Informações de enlace da rede - atribuiu-se para cada aresta do grafo de adjacências, um atraso aleatório, variando de 60 a 100 milissegundos e, então, executado o algoritmo de Dijkstra para encontrar a distância mínima entre cada par de servidores, de onde são definidas as distâncias no primeiro período de tempo. Como estudos de medidas de atraso pela Internet reportam atrasos que variam entre 15 e 300 milissegundos, utilizou-se valores dentro deste intervalo para a criação das instâncias. Definiu-se arbitrariamente que a cada cinco períodos de tempos, um dos enlaces tem seu atraso reconfigurado para um valor entre 60 e 100 ms, e aplicado o algoritmo de Dijkstra novamente, Neves [44].

3. Simetria e assimetria das redes – as instâncias das classes A, B e C possuem redes simétricas onde o tempo para ir de um servidor ao outro é o mesmo que o retorno. As instâncias de classe D possuem canais assimétricos, ou seja, o tempo gasto para ir de um servidor j para um servidor l pode ser diferente do tempo de ir de l a j. Esta assimetria tornam as instâncias de classe D mais próximas da realidade, visto que as redes de computadores também possuem este tipo de características, Neves [44].

4. Espaço em disco - para gerar esta informação de cada servidor foi utilizada a distribuição uniforme de probabilidade com os valores entre 100 e 200 MB para as instâncias de classe A, 100 e 150 GB para a classe B, 3 e 4 GB para a classe C e 2.5 e 3.2 GB para a classe D, Neves [44]. Para as instâncias da classe A, o intervalo para os tamanhos dos discos foi escolhido arbitrariamente tendo em vista que estas instâncias são de teste. Já para as instâncias da classe B, o intervalo foi definido de acordo com a literatura em Zhou et al. [59]. Entretanto, ao perceber que estes valores eram demasiadamente grandes para o número de conteúdos considerados, optou-se por reduzir estes valores para as classes C e D com o objetivo de criar instâncias mais difíceis. Os valores, tanto para a classe C quanto para a classe D foram determinados experimentalmente, Neves [44].

5. Banda dos servidores – a geração das bandas dos servidores foi utilizada a distribuição uniforme de probabilidade com valores oscilando entre 1500 e 2000 MB por período para as instâncias da classe A, 4000 e 4050 MB por período para as classes B e C, e 2300 e 2350 MB por período para a classe D. Os valores de banda para as classes B e C foram escolhidos com base em equipamentos tipo servidores disponíveis no mercado quando as instâncias foram construídas. Contudo, constatou-se posteriormente que os valores de banda para os servidores utilizados nas Classes B

e C geram instâncias em que o PDR é de fácil resolução, e por isso as instâncias da classe D tem o valor da banda dos servidores reduzido para aproximadamente 60% (determinado experimentalmente) dos valores usados nas classes B e C. Os valores usados na classe A foram determinados de maneira arbitrária, Neves [44].

6. Conteúdos dos servidores - para gerar estas informações, a primeira tarefa foi definir o número de conteúdos permanentes, ou seja, o número de conteúdos que existirão desde o primeiro período de tempo até o último período de tempo, o número de conteúdos voláteis, ou seja, que surgem fora do período inicial e podem ser removidos da CDN antes do período final, e também atribuir a cada conteúdo um identificador único. No trabalho de Neves [44] estabeleceu-se que o número de conteúdos permanentes é de 3 conteúdos para as instâncias da classe A, 10 para as instâncias da classe B e C e 12 para as instâncias da classe D. Já para determinar o número de conteúdos voláteis são utilizados números aleatórios que variam entre 1 e 3 para a classe A e entre 1 e 5 para as classes B e C. Para as instâncias da classe D este número é escolhido de maneira aleatória entre 1 e 7, Neves [44].

7. Requisições - para a geração das requisições considerou-se a necessidade de conhecimento das informações dos conteúdos devido ao fato de um número maior de requisições ser dado aos conteúdos populares e também para não gerar requisições para conteúdos que não se encontram disponíveis na CDN. Por conteúdos não disponíveis entenda-se conteúdos que ainda não foram criados ou conteúdos que já foram removidos. Cada requisição também sabe o período em que ela é recebida pela CDN, Neves[44]. Neste contexto, as informações sobre o número de servidores são necessárias, uma vez que, quanto maior o número de servidores, maior o número de pontos de entrada para a CDN e consequentemente maior o número de requisições. Outro motivo para se ter as informações sobre o número de servidores no momento da geração das requisições é o fato de que cada requisição é atrelada a um servidor, chamado servidor de origem, que é o servidor ao qual o cliente que gerou a requisição está conectado. Também são atribuídos atrasos locais às requisições, que representam o tempo que os pacotes levam para transitar entre o cliente e o servidor ao qual o cliente está conectado e para cada requisição é atribuído um servidor aleatoriamente de modo que todos os servidores possuem a mesma chance de escolha.

5.3 Resultados Computacionais do Modelo Exato

Para produzir os resultados computacionais referentes a aplicação do modelo exato na resolução do PDCRR, foram executadas todas as instâncias do problema já implementadas anteriormente por Boechat [18] para as quais foram obtidos os seguintes resultados constantes das Tabelas 5.1, 5.2, 5.3 e 5.4.

Conforme os resultados apresentados, observamos que o modelo exato não foi capaz de resolver as instâncias mais difíceis do PDCRR dentro do tempo estabelecido de 3 horas, parâmetro de tempo constante do modelo já estabelecido em Boechat [18], devido a complexidade do problema para tais instâncias. Estas instâncias estão contidas na Tabela 5.5.

As instâncias listadas na Tabela 5.5 foram o alvo dos testes com a implementação do algoritmo *Local Branch* e do BRKGA, e espera-se que os algoritmos resolvam ou melhorem os resultados onde o modelo exato implementado não foi capaz de resolver. Cabe ressaltar que a informação Status da Solução contida nas tabelas refere-se à classificação da solução utilizada pelo resolvedor CPLEX, as quais: 1) Ótimo - solução ótimo; 2) Ótimo Tol - solução ótima dentro da tolerância da aproximação numérica e 3) Tempo - melhor solução encontrada dentro do tempo limite de 3 horas.

Para as instâncias com 10 servidores o resolvedor não conseguiu encontrar a solução ótima ou comprovar a otimalidade da solução encontrada dentro do tempo estimado de 3 horas nas instâncias 10.17 e 10.19, como pode ser visto na Tabela 5.1.

Tabela 5.1: Resultados Computacionais para o Modelo Exato e Instâncias com 10 servidores.

Servidor.Instância	Tempo	Valor da FO	Status da Solução
10.01	2.7	3.44679e+05	Ótimo
10.02	2.6	3.31019e+05	Ótimo
10.03	2.3	3.07892e+05	Ótimo
10.04	3.4	4.03919e+05	Ótimo
10.05	2.4	3.03442e+05	Ótimo Tol
10.06	42.1	6.78991e+06	Ótimo Tol
10.07	78.3	6.8307e+06	Ótimo Tol
10.08	58.3	6.63729e+06	Ótimo Tol

Tabela 5.1 – *Continuação da tabela*

Servidor.Instância	Tempo	Valor da FO	Status da Solução
10.09	55.4	6.85738e+06	Ótimo Tol
10.10	61.6	7.28641e+06	Ótimo Tol
10.11	34.9	6.78991e+06	Ótimo Tol
10.12	77.2	6.8307e+06	Ótimo Tol
10.13	31.8	6.63729e+06	Ótimo
10.14	49.8	6.85738e+06	Ótimo
10.15	45.8	7.28641e+06	Ótimo
10.16	18233.7	1.37099e+07	Ótimo Tol
10.17	40296.2	1.40279e+07	Tempo
10.18	10631.9	1.37915e+07	Ótimo Tol
10.19	41034.8	1.31696e+07	Tempo
10.20	583.1	1.36326e+07	Ótimo Tol

Para as instâncias com 20 servidores o resolvedor não conseguiu encontrar a solução ótima ou comprovar a otimalidade da solução encontrada dentro do tempo estimado de 3 horas nas instâncias da Classe D: 20.16, 20.17, 20.18, 20.19 e 20.20, como pode ser visto na Tabela 5.2.

Tabela 5.2: Resultados Computacionais para o Modelo Exato e Instâncias com 10 servidores.

Servidor.Instância	Tempo	Valor da FO	Status da Solução
20.01	313.5	5.00032e+05	Ótimo
20.02	28.9	9.27899e+05	Ótimo Tol
20.03	14.8	9.69970e+05	Ótimo
20.04	17.0	7.49350e+05	Ótimo
20.05	21.7	8.33380e+05	Ótimo Tol
20.06	1192.6	1.38259e+07	Ótimo Tol
20.07	425.5	1.29682e+07	Ótimo Tol
20.08	574.4	1.32524e+07	Ótimo Tol
20.09	7899.4	1.3705e+07	Ótimo Tol
20.10	1697.8	1.27435e+07	Ótimo Tol
20.11	1395.9	1.38259e+07	Ótimo Tol

Tabela 5.2 – *Continuação da tabela*

Servidor.Instância	Tempo	Valor da FO	Status da Solução
20.12	403.2	1.29682e+07	Ótimo Tol
20.13	433.3	1.32524e+07	Ótimo Tol
20.14	6666.5	1.3705e+07	Ótimo Tol
20.15	1103.7	1.27435e+07	Ótimo Tol
20.16	33882.2	2.69611e+07	Tempo
20.17	29484.6	2.63159e+07	Tempo
20.18	37732.2	2.71112e+07	Tempo
20.19	35166.5	2.63314e+07	Tempo
20.20	30549.4	2.65977e+07	Tempo

Para as instâncias com 30 servidores o resolvedor não conseguiu encontrar a solução ótima ou comprovar a otimalidade da solução encontrada dentro do tempo estimado de 3 horas nas instâncias da Classe D: 30.16, 30.17, 30.18, 30.19 e 30.20, como pode ser visto na Tabela 5.3.

Tabela 5.3: Resultados Computacionais para o Modelo Exato e Instâncias com 30 servidores.

Servidor.Instância	Tempo	Valor da FO	Status da Solução
30.01	1094.0	1.38738e+06	Ótimo
30.02	999.8	1.2334e+06	Ótimo Tol
30.03	927.5	1.2698e+06	Ótimo
30.04	884.8	1.23047e+06	Ótimo
30.05	1264.5	1.5995e+06	Ótimo Tol
30.06	2892.4	1.93082e+07	Ótimo Tol
30.07	3545.4	1.83383e+07	Ótimo Tol
30.08	3640.0	1.87637e+07	Ótimo Tol
30.09	13692.5	1.85444e+07	Ótimo Tol
30.10	4239.0	1.81754e+07	Ótimo Tol
30.11	2313.8	1.93082e+07	Ótimo Tol
30.12	3523.5	1.83383e+07	Ótimo Tol
30.13	3742.1	1.87637e+07	Ótimo Tol
30.14	30203.2	1.85444e+07	Ótimo Tol

Tabela 5.3 – *Continuação da tabela*

Servidor.Instância	Tempo	Valor da FO	Status da Solução
30.15	4195.4	1.81754e+07	Ótimo Tol
30.16	4806.0	1.3639e+12	Tempo
30.17	3789.3	1.40965e+12	Tempo
30.18	3632.4	1.40304e+12	Tempo
30.19	6183.3	1.62524e+12	Tempo
30.20	4926.9	1.34715e+12	Tempo

Para as instâncias com 50 servidores o resolvedor conseguiu encontrar a solução ótima dentro do tempo estimado de 3 horas apenas nas instâncias da Classe A: 50.01, 50.02, 50.03, 50.04 e 50.05, como pode ser observado na Tabela 5.4.

Tabela 5.4: Resultados Computacionais para o Modelo Exato e Instâncias com 50 servidores.

Servidor.Instância	Tempo	Valor da FO	Status da Solução
50.01	4018.4	2.30765e+06	Ótimo Tol
50.02	2182.6	2.14434e+06	Ótimo Tol
50.03	3517.4	2.47563e+06	Ótimo Tol
50.04	2991.4	2.95794e+06	Ótimo
50.05	3388.7	2.29492e+06	Ótimo
50.06	36072.0	4.05993e+07	Tempo
50.07	12781.2	3.64903e+07	Tempo
50.08	2612.3	3.28104e+07	Tempo
50.09	12738.6	4.50675e+07	Tempo
50.10	4102.6	4.02088e+07	Tempo
50.11	1974.0	4.05059e+07	Tempo
50.12	17294.8	3.64903e+07	Tempo
50.13	3754.0	1.39376e+12	Tempo
50.14	6237.2	1.38716e+12	Tempo
50.15	5685.5	1.23334e+12	Tempo
50.16	16804.5	2.99171e+12	Tempo
50.17	22579.9	2.84913e+12	Tempo

Tabela 5.4 – *Continuação da tabela*

Servidor.Instância	Tempo	Valor da FO	Status da Solução
50.18	15728.0	3.2833e+12	Tempo
50.19	1588.6	3.13084e+12	Tempo
50.20	1452.0	3.1316e+12	Tempo

As instâncias contidas na Tabela 5.5 não foram resolvidas dentro do tempo estabelecido de 3 horas pelo resolvedor CPLEX.

Tabela 5.5: Resultados Computacionais para o Modelo Exato e Instâncias que não foram resolvidos dentro do tempo.

Servidor.Intância	Tempo	Valor da FO
10.17	40296.2	$1.40279e+07$
10.19	41034.8	$1.31696e+07$
20.16	33882.2	$2.69611e+07$
20.17	29484.6	$2.63159e+07$
20.18	37732.2	$2.71112e+07$
20.19	35166.5	$2.63314e+07$
20.20	30549.4	$2.65977e+07$
30.16	4806.0	$1.3639e+12$
30.17	3789.3	$1.40965e+12$
30.18	3632.4	$1.40304e+12$
30.19	6183.3	$1.62524e+12$
30.20	4926.9	$1.34715e+12$
50.06	36072.0	$4.05993e+07$
50.07	12781.2	$3.64903e+07$
50.08	2612.3	$3.28104e+07$
50.09	12738.6	$4.50675e+07$
50.10	4102.6	$4.02088e+07$
50.11	1974.0	$4.05059e+07$
50.12	17294.8	$3.64903e+07$
50.13	3754.0	$1.39376e+12$

Tabela 5.5 – *Continuação da tabela*

Servidor.Intância	Tempo	Valor da FO
50.14	6237.2	$1.38716e+12$
50.15	5685.5	$1.23334e+12$
50.16	16804.5	$2.99171e+12$
50.17	22579.9	$2.84913e+12$
50.18	15728.0	$3.2833e+12$
50.19	1588.6	$3.13084e+12$
50.20	1452.0	$3.1316e+12$

Cabe ressaltar que, quando observamos as instâncias com 10, 20 e 30 servidores, Tabelas 5.1, 5.2 e 5.3, o modelo exato conseguiu encontrar a solução ótima para todas as instâncias das Classes A, B e C. Para as instâncias da Classe D, o modelo exato não conseguiu encontrar a solução ótima ou comprovar a otimalidade da solução encontrada dentro do tempo estimado de 3 horas, com exceção das instâncias 10.16, 10.18 e 10.20.

Para as instâncias com 50 servidores, Tabela 5.4, o tamanho das instâncias realmente influenciou, pois só foram resolvidas as instâncias fáceis da Classe A, nenhumas das Classes B e C, que possuem restrições menos severas que a Classe D, e nem as da Classe D, foram resolvidas, como pode ser observado na Tabela 5.5.

5.4 Resultados Computacionais da Implementação do *Local Branch*

Para validar o algoritmo implementado no presente trabalho foram realizados testes comparativos entre o código com a técnica do *Local Branch* e os resultados obtidos com modelo exato, com todas as instâncias resolvidas dentro do limite de tempo estabelecido, as quais foram encontradas suas soluções ótimas, em que a comparação dos resultados permitiu concluir que o código estava pronto para produzir os resultados do trabalho. Os resultados dos testes de validação do código foram feitos utilizando como parâmetro de tempo para encontrar a solução inicial em 1% do tempo de execução proposto no modelo exato e uma variação de 50% das variáveis que compõem o suporte binário, os resultados podem ser vistos nas Tabelas 5.6, 5.7, 5.8 e 5.9. Cabe ressaltar que as colunas das tabelas cujo cabeçalho é Etapa LB, refere-se a qual etapa do código a solução foi encontrada para a instância, podendo ser esta etapa (1), se a solução fora encontrada pela função

solver_i, (2) se a solução fora encontrada pela função local_branch ou (3) caso a solução fora encontrada pela função solver_f.

Pode-se observar que para as instâncias com 10 servidores, conforme a Tabela 5.6, o código *local branch* encontrou a solução ótima para todas as instâncias. Devido as instâncias com 10 servidores possuírem um número menor de variáveis, a função inicial conseguiu encontrar a solução ótima na maioria das instâncias e, consequentemente, teve um gasto de tempo similar ao modelo exato, destoando apenas onde a solução não foi encontrada pela função inicial, instâncias 10.16, 10.18 e 10.20.

Tabela 5.6: Validação do Código *Local Branch* para Instâncias com 10 servidores.

Serv.Int	Tempo Mod Exato	Tempo LB 50%	*IND*	Etapa LB
10.01	2.7	2.9	0.00%	(1)
10.02	2.6	2.8	0.00%	(1)
10.03	2.3	2.5	0.00%	(1)
10.04	3.4	3.9	0.00%	(1)
10.05	2.4	2.8	0.00%	(1)
10.06	42.2	41.1	0.00%	(1)
10.07	78.3	80.7	0.00%	(1)
10.08	58.4	53.2	0.00%	(1)
10.09	55.4	51.7	0.00%	(1)
10.10	61.6	58.1	0.00%	(1)
10.11	34.9	34.1	0.00%	(1)
10.12	77.2	80.4	0.00%	(1)
10.13	31.8	34.6	0.00%	(1)
10.14	49.8	55.3	0.00%	(1)
10.15	45.8	46.2	0.00%	(1)
10.16	18233.7	49180.8	0.00%	(2)
10.18	10631.9	24990.1	0.00%	(2)
10.20	583.1	3607.9	0.00%	(2)

Pode-se observar que para as instâncias com 20 servidores, conforme a Tabela 5.7, o código *local branch* encontrou a solução ótima para todas as instâncias. Diferentemente das instâncias com 10 servidores, apenas para as instâncias da Classe A, a solução ótima

foi encontrada pela função inicial, sendo as demais pela função *local branch*. Com relação ao tempo, as instâncias cujas soluções ótimas foram entradas pela função inicial, instâncias da Classe A, o gasto de tempo foi similar ao modelo exato, para as instâncias das Classes B e C a demanda de tempo foi maior.

Tabela 5.7: Validação do Código *Local Branch* para Instâncias com 20 servidores.

Serv.Int	Tempo Mod Exato	Tempo LB 50%	*IND*	Etapa LB
20.01	313.5	308.7	0.00%	(1)
20.02	28.9	28.6	0.00%	(1)
20.03	14.8	14.7	0.00%	(1)
20.04	17.0	16.9	0.00%	(1)
20.05	21.7	21.7	0.00%	(1)
20.06	1192.6	9497.0	0.00%	(2)
20.07	425.5	7256.8	0.00%	(2)
20.08	574.4	9671.1	0.00%	(2)
20.09	7899.4	56406.6	0.00%	(2)
20.10	1697.8	13795.6	0.00%	(2)
20.11	1395.9	9524.2	0.00%	(2)
20.12	403.2	5990.2	0.00%	(2)
20.13	433.3	10410.7	0.00%	(2)
20.14	6666.5	52893.0	0.00%	(2)
20.15	1103.7	12753.5	0.00%	(2)

Para as instâncias com 30 servidores, conforme a Tabela 5.8, o código *local branch* encontrou maior dificuldade para encontrar a solução ótima para todas as instâncias, demandando mais tempo, sendo que para as instâncias da Classe C, embora tenha encontrado a solução ótima, comparado como a solução do modelo exato, não houve tempo para comprovar sua otimalidade.

Tabela 5.8: Validação do Código *Local Branch* para Instâncias com 30 servidores.

Serv.Int	Tempo Mod Exato	Tempo LB 50%	*IND*	Etapa LB
30.01	1094.0	1434.2	0.00%	(2)
30.02	999.8	3162.0	0.00%	(2)
30.03	927.5	485.9	0.00%	(2)
30.04	884.8	2155.9	0.00%	(2)
30.05	1264.5	3947.9	0.00%	(2)
30.06	2892.4	36303.7	0.00%	(2)
30.07	3545.4	42682.1	0.00%	(3)
30.08	3640.0	56478.7	0.00%	(2)
30.09	13692.5	54729.4	0.00%	(2)
30.10	4239.0	53323.0	0.00%	(2)
30.11	2313.8	16048.0	0.00%	*
30.12	3523.5	42642.8	0.00%	*
30.13	3742.1	58367.7	0.00%	*
30.14	30203.2	57364.5	0.00%	*
30.15	4195.4	54477.9	0.00%	*
* Excedeu o tempo e não comprovou a otimalidade da solução				

Para as instâncias com 50 servidores, conforme a Tabela 5.9, o código *local branch* encontrou a solução ótima para todas as instâncias, com uma demanda maior de tempo.

Tabela 5.9: Validação do Código *Local Branch* para Instâncias com 50 servidores.

Serv.Int	Tempo Mod Exato	Tempo LB 50%	*IND*	Etapa LB
50.01	4018.4	9146.2	0.00%	(2)
50.02	2182.6	8447.9	0.00%	(2)
50.03	3517.4	16164.7	0.00%	(2)
50.04	2991.4	9190.7	0.00%	(2)
50.05	3388.7	9553.2	0.00%	(3)

Durante as etapas de implementação do algoritmo *local branch* foi necessário definir uma estratégia para escolher o tamanho da vizinhança que melhor se adaptaria ao problema, esta escolha poderia ser feita por um número estimado e fixo, porém, devido a dimensão do problema e sua grande quantidade de variáveis, optou-se por fazer uma análise em relação a um percentual das variáveis do suporte binário que poderiam permutar seu valor, ou seja, procurou-se estabelecer um percentual do número de variáveis binárias que assumiam valores unitários a partir de uma solução inicial, que poderiam permutar com as variáveis binárias que assumiram o valor zero nesta solução, definido o tamanho da vizinhança. Neste sentido, o algoritmo apresentou melhores tempos de resolução do problema quando se definia uma vizinhança de tamanho médio, entre 50% e 70% da quantidade de variáveis do suporte binário, tamanho para o qual estas vizinhanças foram resolvidas rapidamente pelo CPLEX, o que podemos observar na Tabela 5.10

Tabela 5.10: Resultados Computacionais para o *Local Branch* com Instâncias da Classe A

% Sup Bin	10-5	20-5	30-5	50-5
5	27.5	42757.4	5996.15	13099.0
10	19.9	41727.2	5869.35	16380.9
20	15.8	1313.6	4993.28	10505.4
30	13.3	1231.8	3766.32	9355.6
40	12.1	1126.8	3853.59	**8906.6**
50	12.0	882.5	3736.5	9781.8
60	**10.6**	**876.6**	3658.0	9682.1
70	12.0	984.7	**3637.5**	9723.5
80	13.9	1027.04	3939.9	9089.8
90	11.7	915.722	3605.8	9451.4
95	11.9	1028.39	3741.6	10721.7

Embora o algoritmo tenha apresentado os resultados contidos na Tabela 5.10 para as instâncias da Classe A, quando aplicado em instâncias mais complexas do modelo proposto no trabalho, instâncias da Classe D, o algoritmo não apresentou o mesmo comportamento e os melhores tempos foram obtidos quando o percentual de permuta das variáveis do suporte binário era baixo, próximo a 1%, indicando que nestas situações, vizinhanças menores são resolvidas mais facilmente pelo CPLEX, e levando a menores tempos para a resolução do problema.

Para realizar o teste mencionado no parágrafo anterior, foi escolhida uma instância da classe D que foi resolvida dentro do tempo pelo método exato, proporcionando um parâmetro temporal de referência real que poderia ser utilizado para comparação, pois para outras instâncias da classe D não seria possível comparar devido ao fato de que o resolvedor não logrou êxito na resolução do problema dentro do limite do tempo. Observando os resultados do modelo exato, somente as instância 10.16, 10.18 e 10.20, da classe D, foram resolvidas dentro do tempo, e para o teste foi escolhido arbitrariamente a instância 10.18 para o teste que está apresentado na Tabela 5.11.

Tabela 5.11: Resultados Computacionais com *Local Branch* para Análise do tempo de resolução variando a vizinhança do suporte binário.

% Sup Bin	Tempo
1	26713.6
5	30740.7
10	33799.2
20	36748.4
30	39558.0
40	37585.2
50	44011.5
60	53152.5
70	52807.4
80	57429.7
90	59497.4
95	60122.3

O teste realizado apresentou duas dificuldades, a primeira foi a necessidade em definir o quão pequena deveria ser a vizinhança, pois os testes demonstraram que vizinhanças muito pequenas tem a vantagem de ser resolvidas dentro do tempo pelo resolvedor, porém algoritmo caminha lentamente para a solução ótima, sendo necessário resolver muitas pequenas vizinhanças até se aproximar da possível solução ótima do problema ou de um ótimo local, demandando muito tempo e possibilitando que o algoritmo, ao permitir poucas permutações das variáveis do suporte binário ficasse preso a uma solução ótima local de baixa qualidade e não explorando boa parte do espaço viável, ou seja, a vizinhança deve ser pequena o suficiente para facilitar que o resolvedor possa encontrar uma solução

ótima dentro da vizinhança em um tempo adequado e com um tamanho que permita o algoritmo percorra uma boa parte do espaço de soluções viáveis em busca da solução ótima do problema, evitando de ficar preso a um ótimo local de baixa qualidade. A segunda dificuldade foi encontrar um forma de definir uma boa solução inicial que não estivesse tão distante da solução ótima do problema, com isso a qualidade da solução inicial tornou-se fundamental para iniciar o algoritmo.

Para satisfazer a primeira dificuldade foi observado nos arquivos com as saídas das soluções do teste realizado com a instância 10.18 a distância entre valor da F.O. da solução apresentada pela função solver_i, a solução apresentada dentro da função local_branch e a solução ótima do problema. Analisando os aspectos citados com as variações dos tamanhos das vizinhanças, conforme consta da Tabela 5.12, chegou-se a conclusão que o tamanho adequado para o teste com as instâncias que não foram resolvidas dentro do tempo pelo modelo exato seria de 50%. Apesar de possuir um tempo de resolução maior, a função local_branch se aproximou da solução ótima dentro da mesma ordem de grandeza, ou seja, na ordem de $10e+07$ e, possivelmente, caso a função local_branch não encontre a solução ótima, percorreria grande parte do espaço de busca, diminuindo o tamanho do espaço restante e facilitando que a solução fosse encontrada pela função solver_f, que busca a solução no espaço viável que não foi explorado pelo algoritmo.

Tabela 5.12: Resultados Computacionais com o Local Branch para Análise dos valores das F.O.

Instancia 10-18					
% Sup Bin	Fun Obj	Tempo	Status Sol	Fun Obj LB	Status LB
1	1.37915e+07	26713.6	Ótimo Tol	2.51155e+11	Tempo
5	1.37915e+07	30740.7	Ótimo Tol	2.12743e+11	Tempo
10	1.37915e+07	33799.2	Ótimo Tol	1.39985e+11	Tempo
20	1.37915e+07	36748.4	Ótimo Tol	6.25212e+10	Tempo
30	1.37915e+07	39558.0	Ótimo Tol	1.07215e+10	Tempo
40	1.37915e+07	37585.2	Ótimo Tol	5.32171e+08	Tempo
50	1.37915e+07	44011.5	Ótimo Tol	2.32631e+07	Tempo
60	1.37915e+07	53152.5	Ótimo Tol	1.53976e+07	Tempo
70	1.37915e+07	52807.4	Ótimo Tol	1.50867e+07	Tempo
80	1.37915e+07	57429.7	Ótimo Tol	1.48592e+07	Tempo
90	1.37915e+07	59497.4	Ótimo Tol	1.46792e+07	Tempo

Tabela 5.12 – *Continuação da tabela*

Instancia 10-18					
% Sup Bin	Fun Obj	Tempo	Status Sol	Fun Obj LB	Status LB
95	1.37915e+07	60122.3	Ótimo Tol	1.45972e+07	Tempo

A outra dificuldade apresentada e que Fischestti [25] destaca como sendo relevante é a qualidade da solução viável inicial, na qual, ainda no início da computação, possibilite que soluções que estejam a uma distância aceitável da solução ótima sejam encontradas num limite de tempo menor. Inicialmente havia sido implementado na função solver_i, que encontra a solução inicial no código implementado, o seguinte comando do CPLEX:

CPXsetlongparam (cpxEnv, CPXPARAM_MIP_Limits_Solutions, 1)

O parâmetro CPXPARAM_MIP_Limits_Solutions controla o número máximo de solução incumbente a ser encontrada antes de parar a otimização e configurando este parâmetro com o valor 1, o CPLEX estava encontrando a primeira solução viável para servir como solução inicial. Neste contexto foi observado nos testes com algumas instância que a qualidade desta solução era péssima, ou seja, muito distante da solução ótima. Então ao invés de estabelecer um número limite de soluções, optou-se por estabelecer um tempo razoavelmente grande na expectativa de gastar um tempo maior no início da computação e encontrar uma solução inicial com boa qualidade e que estivesse o mais próxima possível da solução ótima, de modo que permitisse ao resolvedor, diante da complexidade do problema, resolver o modelo do PDCRR utilizando a técnica proposta por Fischetti et al. [25]. O tempo fornecido para encontrar a solução inicial foi de 30% do tempo destinado para o resolvedor e o CPLEX foi configurado com o seguinte parâmetro na função solver_i:

CPXsetdblparam (cpxEnv, CPX_PARAM_TILIM, 0.3*runningTime)

A comparação entre as soluções iniciais utilizando os dois parâmetros citados acima pode ser vista na Tabela 5.13.

Após as observações mencionadas acima, que foram obtidas dos testes, configurou-se o código para a tentativa de solucionar as instâncias que não foram resolvidas dentro do limite do tempo pelo método exato e contida na Tabela 5.5 da seguinte forma:

a) solver_i – com 30% do tempo limite estabelecido para o método exato;

b) permuta de 50% das variáveis do suporte binário; e

c) tempo de execução do loop da função local_branch de 150% do tempo de resolução do método exato. Este parâmetro foi escolhido arbitrariamente de modo a permitir que, caso o tempo de limite fosse ultrapassado pela primeira vizinhança, o código teria uma segunda possibilidade de explorar uma outra vizinhança.

Uma estimativa do tempo para cada função do código, considerando t = 3 horas, é até 0.3t para a função solver_i, 1.5t para o looping da função local_branch e 1.0t para a função solver_f, podendo o código utilizar um tempo total de até 2.8t. O tempo é um dos grandes óbices do método quando aplicado no problema estudado no presente trabalho, uma vez que o código pode demorar até 2.8 vezes o tempo de 3 horas. Este fato advém da forma como o algoritmo foi concebido e a complexidade do problema, sendo necessário executar várias funções que manipulam o resolvedor CPLEX, quais sejam, a função solver_i, que configura o resolvedor CPLEX para encontrar a primeira solução do problema, a qual servirá como ponto de partida para definir os conjuntos B, G e C, (3.2)-(3.6), iniciar o método, encontrando um suporte binário e definindo a primeira vizinhança; a função sup_binario, que captura os índices das variáveis de uma determinada solução cujos valores assumiram o valor um, definido desta forma o suporte binário e consequentemente permitindo a geração de uma vizinhança; a função local_branch, que utiliza a informação dos índices capturados pela função sup_binario e acrescenta uma nova linha de restrição ao problema, definindo uma vizinhança de busca e utiliza o resolvedor CPLEX para encontrar a solução dentro da vizinhança respeitando as restrições do problema; a função complem_lb que limita o espaço de busca excluindo a vizinhança já explorada pela função local_branch, evitando que buscas em espaços já explorados sejam realizadas; e a função solver_f, que é chamada quando o código não consegue melhorias nas soluções utilizando a técnica do *local branch* utilizando o CPLEX ou extrapola o tempo e explora o restante do espaço de busca que não foi explorado pelas vizinhanças.

Na Tabela 5.13 estão os resultados apresentados pelo código aplicado nas instâncias que não foram resolvidas dentro do tempo pelo método exato referentes a função que fornece a solução inicial para o código e o suporte binário inicial. Observa-se nesta tabela que a qualidade da solução inicial é melhor quando a função que encontra a solução inicial utiliza 30% do tempo para encontrar uma solução.

Tabela 5.13: Comparação entre as soluções iniciais utilizando os dois parâmetros:

Serv.Int	FO Solver_i com 1ª Sol do CPLEX	FO Solver_i com 30% tempo
10.17	4.36483e+11	1.40279e+07
10.19	5.0221e+11	1.31699e+07
20.16	1.02461e+12	2.69683e+07
20.17	1.12755e+12	2.6319e+07
20.18	1.09477e+12	2.71265e+07
20.19	1.00224e+12	2.63445e+07
20.20	1.01182e+12	2.66057e+07
30.16	1.68668e+12	1.3639e+12
30.17	1.78373e+12	1.40965e+12
30.18	1.7541e+12	1.40304e+12
30.19	2.02381e+12	1.62524e+12
30.20	1.69475e+12	1.34715e+12
50.07	**1.67765e+12**	**1.67765e+12**
50.08	1.61505e+12	4.39056e+07
50.09	1.75987e+12	4.50675e+07
50.10	1.6379e+12	4.02088e+07
50.11	**1.81868e+12**	**1.81868e+12**
50.12	**1.67765e+12**	**1.67765e+12**
50.13	1.63888e+12	1.39376e+12
50.14	1.77963e+12	1.38716e+12
50.15	1.65287e+12	1.23334e+12
50.16	3.53881e+12	2.99171e+12
50.17	3.34157e+12	2.84913e+12
50.18	3.86412e+12	3.2833e+12
50.19	3.68076e+12	3.13084e+12
50.20	3.64832e+12	3.1316e+12

Tabela 5.14: Resultados Computacionais com Local Branch para as Instâncias não Resolvidas dentro do tempo pelo Método Exato e vizinhança permutando até 50% do suporte binário

Serv.Int	FO Mod Exato	Tempo	FO LB k50%	Tempo	*IND*
10.17	1.40279e+07	40296.0	1.40279e+07	91425.0	0.00%
10.19	1.31696e+07	41035.0	1.31696e+07	93431.0	0.00%
20.16	2.69611e+07	33882.0	2.69605e+07	81403.0	0.00%
20.17	2.63159e+07	29485.0	2.63158e+07	61965.0	0.00%
20.18	2.71112e+07	37732.0	2.71153e+07	75721.0	0.02%
20.19	2.63314e+07	35167.0	2.63333e+07	76616.0	0.01%
20.20	2.65977e+07	30549.0	2.65877e+07	68568.0	-0.04%
30.16	1.3639e+12	4806.0	1.3639e+12	19060.0	0.00%
30.17	1.40965e+12	3789.0	1.40965e+12	18068.0	0.00%
30.18	1.40304e+12	3632.0	1.40304e+12	21410.0	0.00%
30.19	1.62524e+12	6183.0	1.62524e+12	22964.0	0.00%
30.20	1.34715e+12	4927.0	1.34715e+12	22107.0	0.00%
50.7	3.64903e+07	12781.0	3.64903e+07	44714.0	0.00%
50.8	3.28104e+07	2612.0	4.39056e+07	16358.0	33.82%
50.9	4.50675e+07	12739.0	4.50675e+07	21322.0	0.00%
50.10	4.02088e+07	4103.0	3.78417e+07	35966.0	-5.89%
50.11	4.05059e+07	1974.0	4.05059e+07	73962.0	0.00%
50.12	3.64903e+07	17295.0	3.64903e+07	63213.0	0.00%
50.13	1.39376e+12	3754.0	1.39376e+12	19927.0	0.00%
50.14	1.38716e+12	6237.0	1.11023e+12	30096.0	-19.96%
50.15	1.23334e+12	5685.0	1.23334e+12	18774.0	0.00%
50.16	2.99171e+12	16805.0	2.99171e+12	19056.0	0.00%
50.17	2.84913e+12	22580.0	2.84913e+12	23850.0	0.00%
50.18	3.2833e+12	15728.0	3.2833e+12	39228.0	0.00%
50.19	3.13084e+12	1589.0	3.13084e+12	30414.0	0.00%
50.20	3.1316e+12	1452.0	3.1316e+12	34738.0	0.00%

Após estes testes com as instâncias que não foram resolvidas dentro do tempo pelo Modelo Exato e com *Local Branching* configurado para permutar 50% das variáveis do

suporte binário, observou-se melhoras em 03 resultados comparados com os apresentados com o modelo exato, as mais significativas na instância 50.10 e 50.14. Diante dos resultados e do fato de o resolvedor CPLEX não dava conta de encontrar a solução ótima dentro do tempo estabelecido para a busca da solução ótima dentro da vizinhança haja visto a complexidade do problema. Optou-se então em realizar os testes reduzindo o tamanho da vizinhança e permitindo apenas a permuta de 1% das variáveis binárias dentro do suporte binário, conforme a Tabela 5.15.

Tabela 5.15: Resultados Computacionais com *Local Branch* para as Instâncias não Resolvidas dentro do tempo pelo Método Exato e vizinhança permutando até 1% do suporte binário

Serv.Int	FO Mod Exato	Tempo	FO LB k50%	Tempo	*IND*
10.17	1.40279e+07	40296.0	1.40279e+07	72567.0	0.00%
10.19	1.31696e+07	41035.0	1.31696e+07	75500.0	0.00%
20.16	2.69611e+07	33882.0	2.69635e+07	79056.0	0.01%
20.17	2.63159e+07	29485.0	2.63136e+07	66201.2	-0.01%
20.18	2.71112e+07	37732.0	2.71175e+07	77313.4	0.02%
20.19	2.63314e+07	35167.0	2.6334e+07	60098.7	0.01%
20.20	2.65977e+07	30549.0	2.65923e+07	70790.9	-0.02%
30.16	1.3639e+12	4806.0	1.36195e+12	21523.2	-0.14%
30.17	1.40965e+12	3789.0	1.40098e+12	18949.9	-0.62%
30.18	1.40304e+12	3632.0	1.40161e+12	21872	-0.10%
30.19	1.62524e+12	6183.0	1.62398e+12	24036.5	-0.08%
30.20	1.34715e+12	4927.0	1.33777e+12	21050.2	-0.70%
50.7	3.64903e+07	12781.0	3.64903e+07	28086.6	0.00%
50.8	3.28104e+07	2612.0	4.35296e+07	34918	32.67%
50.9	4.50675e+07	12739.0	4.50675e+07	22239.2	0.00%
50.10	4.02088e+07	4103.0	4.02088e+07	21371.8	0.00%
50.11	4.05059e+07	1974.0	4.05059e+07	77483.2	0.00%
50.12	3.64903e+07	17295.0	3.61945e+07	42558.9	-0.81%
50.13	1.39376e+12	3754.0	1.39376e+12	19344	0.00%
50.14	1.38716e+12	6237.0	1.38716e+12	23287	0.00%
50.15	1.23334e+12	5685.0	1.0765e+12	36019.1	-12.72%

Tabela 5.15 – *Continuação da tabela*

Serv.Int	FO Mod Exato	Tempo	FO LB k50%	Tempo	*IND*
50.16	2.99171e+12	16805.0	2.99171e+12	24063.4	0.00%
50.17	2.84913e+12	22580.0	2.84913e+12	16764.8	0.00%
50.18	3.2833e+12	15728.0	3.2833e+12	43498.9	0.00%
50.19	3.13084e+12	1589.0	3.13084e+12	29141.4	0.00%
50.20	3.1316e+12	1452.0	3.1316e+12	30240.5	0.00%

Com o novo teste realizado conseguiu-se pequenas melhoraras em 08 resultados comparados com os apresentados com o modelo exato, a mais significativa na instância 50.15.

Então comparou-se os resultados das instâncias da Tabela 5.5 entre o modelo exato, o *local branch* com a vizinhança permitindo 50% de permutas das variáveis do suporte binário, Tabela 5.14 e o *Local Branch* com a vizinhança permitindo 1% de permutas das variáveis dos suporte binário, Tabela 5.15.

Observa-se na Tabela 5.16 que, com relação às instâncias que não foram resolvidas dentro do tempo, o modelo exato obteve melhor desempenho em apenas (03) três instâncias, o algorítmo *local branch* obteve melhor desempenho em (12) doze instâncias e empatou com o modelo exato nas demais, tendo como desvantagem o gasto maior com o tempo.

Tabela 5.16: Comparação dos Resultados Computacionais entre o Modelo Exato e o Local Branch com vizinhança permutando até 50% e 1% do suporte binário.

Serv.Int	Pior Resultado	Melhor Resultado
10.17	Empate	
10.19	Empate	
20.16	LB K 1%	LB K 50%
20.17	LB K 50%	LB K 1%
20.18	LB K 1%	Mod Exato
20.19	LB K 1%	Mod Exato
20.20	LB K 1%	LB K 50%
30.16	Mod Exato e LB K 50%	LB K 1%
30.17	Mod Exato e LB K 50%	LB K 1%

Tabela 5.16 – *Continuação da tabela*

Serv.Int	Pior Resultado	Melhor Resultado
30.18	Mod Exato e LB K 50%	LB K 1%
30.19	Mod Exato e LB K 50%	LB K 1%
30.20	Mod Exato e LB K 50%	LB K 1%
50.6	Empate	
50.7	Empate	
50.8	LB K 50%	Mod Exato
50.9	Empate	
50.10	Mod Exato e LB K 1%	LB K 50%
50.11	Empate	
50.12	Mod Exato e LB K 50%	LB K 1%
50.13	Empate	
50.14	Mod Exato e LB K 1%	LB K 50%
50.15	Mod Exato e LB K 50%	LB K 1%
50.16	Empate	
50.17	Empate	
50.18	Empate	
50.19	Empate	
50.20	Empate	

5.5 Resultados Computacionais da Implementação do BRKGA

Para gerar os resultados computacionais utilizando a implementação do BRKGA foram realizados testes, obtendo-se as seguintes configurações na API disponibilizada por Gonçalves et al. [30]:

- Tamanho da população -> n = 100
- Classe Elite -> pe = 0.10n
- Mutação -> pm = 0.1n
- Recombinação -> rhoe = 0.80n

Para que o algoritmo não executasse indefinidamente, foi estabelecido como condição de parada os seguintes parâmetros:

- Número fixo de 100 gerações desde que houve a última melhora na qualidade da melhor solução;
- Tempo máximo de execução de 03 horas;
- Encontrar uma solução com *fitness* igual ao valor obtido pelo Modelo Exato, cuja otimalidade tenha sido comprovada. Este critério, embora não seja recomendado, foi utilizado nas instâncias cujas soluções ótimas eram conhecidas, como forma de economia de tempo para conclusão da pesquisa.

Para validar o BRKGA no presente trabalho foram realizados testes comparativos com os resultados obtidos com modelo exato, com todas as instâncias resolvidas dentro do tempo de 3 horas. Os resultados da validação do BRKGA podem ser observados nas Tabelas 5.17, 5.19, 5.21 e 5.23, onde nas colunas BRKGA, Tempo e Nr Iter, constam a médias geradas com dez sementes aleatórias diferentes para cada instância.

Pode-se observar na Tabela 5.17 que a distância entre a solução ótima encontrada pelo Modelo Exato e a heurística BRKGA não ultrapassou em 5% em nenhuma instância, cabendo ressaltar dois aspectos: o primeiro é que para as instâncias da Classe A a heurística obteve um valor abaixo de 0,1% de distância da solução ótima do problema e que o tempo utilizado pela heurística, principalmente nas Classes B, C e D, foi o limite estabelecido de 3 horas.

Tabela 5.17: Validação da heurística BRKGA para 10 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
10.1	3.44679e+05	2.7	3.44679e+05	579.0	357	0.00%
10.2	3.31019e+05	2.6	3.31172e+05	791.0	537	0.05%
10.3	3.07892e+05	2.3	3.08038e+05	686.0	414	0.05%
10.4	4.03919e+05	3.4	4.04020e+05	1075.0	652	0.02%
10.5	3.03442e+05	2.4	3.03648e+05	451.0	400	0.07%
10.6	6.789910e+06	42.2	7.011706e+06	10804.0	1905	3.27%

Tabela 5.17 – *Continuação da tabela*

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
10.7	6.8307e+06	78.3	7.000978e+06	10802.0	1803	2.49%
10.8	6.63729e+06	58.4	6.908396e+06	10803.0	2033	4.08%
10.9	6.85738e+06	55.4	7.078071e+06	10803.0	1967	3.22%
10.10	7.28641e+06	61.6	7.586374e+06	10803.0	1926	4.12%
10.11	6.78991e+06	34.9	7.013652e+06	10803.0	1889	3.30%
10.12	6.8307e+06	77.2	7.00016e+06	10803.0	1813	2.48%
10.13	6.63729e+06	31.8	6.910393e+06	10803.0	2058	4.11%
10.14	6.85738e+06	49.8	7.081311e+06	10803.0	1961	3.27%
10.15	7.28641e+06	45.8	7.59952e+06	10803.0	1926	4.30%
10.16	1.37099e+07	18233.7	1.3968383e+07	10804.0	1547	1.89%
10.18	1.37915e+07	10631.9	1.4146962e+07	10803.0	1549	2.58%
10.20	1.36326e+07	583.1	1.3886347e+07	10803.0	1762	1.86%

Na Tabela 5.18 são apresentadas a comparação entre o modelo exato e os melhores resultados obtidos pela heurística BRKGA com relação à média dos resultados para as instâncias com 10 servidores.

Tabela 5.18: Melhores resultados da heurística BRKGA para 10 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
10.1	3.44679e+05	2.7	3.44679e+05	432.4	264	0.00%
10.2	3.31019e+05	2.6	3.31019e+05	632.8	430	0.00%
10.3	3.07892e+05	2.3	3.07892e+05	536.4	325	0.00%
10.4	4.03919e+05	3.4	4.03935e+05	1419.6	859	0.00%
10.5	3.03442e+05	2.4	3.03555e+05	371.6	330	0.04%
10.6	6.78991e+06	42.2	6.990409e+06	10804.2	1886	2.95%
10.7	6.8307e+06	78.3	6.979965e+06	10801.5	1784	2.19%
10.8	6.63729e+06	58.4	6.883586e+06	10803.4	2032	3.71%
10.9	6.85738e+06	55.4	7.059194e+06	10800.6	1970	2.94%
10.10	7.28641e+05	61.6	7.565187e+06	10803.6	1932	3.83%
10.11	6.78991e+06	34.9	6.99218e+06	10805.5	1882	2.98%

Tabela 5.18 – *Continuação da tabela*

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
10.12	6.8307e+06	77.2	6.979965e+06	10805.2	1803	2.19%
10.13	6.63729e+06	31.9	6.869082e+06	10801.0	2063	3.49%
10.14	6.85738e+06	49.8	7.066751e+06	10803.5	1956	3.05%
10.15	7.28641e+06	45.8	7.572374e+06	10801.5	1928	3.92%
10.16	1.37099e+07	18233.7	1.3948305e+07	10804.7	1543	1.74%
10.18	1.37915e+07	10631.9	1.4119861e+07	10806.5	1552	2.38%
10.20	1.36326e+07	583.1	1.3871982e+07	10802.7	1540	1.76%

Pode-se observar na Tabela 5.19 que a distância entre a solução ótima encontrada pelo Modelo Exato e a heurística BRKGA para as instâncias da Classe A não ultrapassou 0.1% e para as Classes B e C a distância ficou próxima de 5% da solução ótima, destoando apenas nas instâncias 20.8 e 20.13, as quais ficaram próximas de 13%. Para as Classes B e C o código utilizou o limite estabelecido de 3 horas.

Tabela 5.19: Validação da heurística BRKGA para 20 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
20.1	5.00032e+05	313.5	5.00122e+05	1303.0	473	0.02%
20.2	9.27899e+05	28.9	9.28186e+05	7488.0	943	0.03%
20.3	9.6997e+05	14.8	9.70195e+05	5499.0	820	0.02%
20.4	7.4935e+05	17.0	7.49856e+05	4975.0	870	0.07%
20.5	8.3338e+05	21.7	8.33641e+05	3649.0	633	0.03%
20.6	1.38259e+07	1192.6	1.4826055e+07	10813.0	537	7.23%
20.7	1.29682e+07	425.5	1.3646118e+07	10811.0	524	5.23%
20.8	1.32524e+07	574.4	1.5009949e+07	10812.0	403	13.26%
20.9	1.3705e+07	7899.4	1.4613603e+07	10814.0	557	6.63%
20.10	1.27435e+07	1697.8	1.3429733e+07	10808.0	593	5.38%
20.11	1.38259e+07	1395.9	1.4800647e+07	10810.0	601	7.05%
20.12	1.29682e+07	403.2	1.3652321e+07	10810.0	538	5.28%
20.13	1.32524e+07	433.3	1.5001268e+07	10809.0	562	13.20%
20.14	1.3705e+07	6666.5	1.4607612e+07	10810.0	542	6.59%

Tabela 5.19 – *Continuação da tabela*

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
20.15	1.27435e+07	1103.8	1.3416382e+07	10807.0	526	5.28%

Na Tabela 5.20 são apresentadas a comparação entre o modelo exato e os melhores resultados obtidos pela heurística BRKGA com relação à média dos resultados para as instâncias com 20 servidores.

Tabela 5.20: Melhores resultados da heurística BRKGA para 20 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
20.1	5.00032e+05	313.5	5.00032e+05	970.5	369	0.00%
20.2	9.27899e+05	28.9	9.28037e+05	7125.4	950	0.01%
20.3	9.6997e+05	14.8	9.69971e+05	5718.1	851	0.00%
20.4	7.4935e+05	17.0	7.49350e+05	5976.5	1022	0.00%
20.5	8.3338e+05	21.7	8.33513e+05	3950.3	711	0.02%
20.6	1.38259e+07	1192.6	1.4761499e+07	10810.1	549	6.77%
20.7	1.29682e+07	425.5	1.3612537e+07	10807.1	526	4.97%
20.8	1.32524e+07	574.4	1.4889447e+07	10809.9	581	12.35%
20.9	1.3705e+07	7899.4	1.4562599e+07	10813.2	558	6.26%
20.10	1.27435e+07	1697.8	1.3400914e+07	10801.5	593	5.16%
20.11	1.38259e+07	1395.9	1.4742126e+07	10814.1	624	6.63%
20.12	1.29682e+07	403.2	1.3609974e+07	10801.0	570	4.95%
20.13	1.32524e+07	433.3	1.4801239e+07	10810.2	561	11.69%
20.14	1.3705e+07	6666.5	1.4578782e+07	10816.3	540	6.38%
20.15	1.27435e+07	1103.8	1.3371431e+07	10813.6	583	4.93%

Pode-se observar na Tabela 5.21 que a distância entre a solução ótima encontrada pelo Modelo Exato e a heurística BRKGA para as instâncias da Classe A não ultrapassou 0.3% e para as Classes B e C a distância ficou próxima de 8% da solução ótima, destoando apenas nas instâncias 30.6 e 30.11, as quais ficaram próximas de 10%. Cabe ressaltar que, comparativamente às instâncias com 10 e 20 servidores, a heurística teve um desempenho um pouco inferior provavelmente por se tratar de instâncias maiores. Para todas as classes o código utilizou o limite estabelecido de 3 horas.

Tabela 5.21: Validação da heurística BRKGA para 30 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
30.1	1.38738e+06	1094.0	1.389559e+06	10807.0	766	0.16%
30.2	1.2334e+06	999.8	1.23437e+06	10277.0	1139	0.08%
30.3	1.2698e+06	927.5	1.271568e+06	10804.0	936	0.14%
30.4	1.23047e+06	884.8	1.233506e+06	10774.0	948	0.25%
30.5	1.5995e+06	1264.5	1.603521e+06	10806.0	796	0.25%
30.6	1.93082e+07	2892.4	2.1267024e+07	10823.0	269	10.15%
30.7	1.83383e+07	3545.4	1.9902207e+07	10817.0	289	8.53%
30.8	1.87637e+07	3640.0	2.0309215e+07	10817.0	268	8.24%
30.9	1.85444e+07	13692.5	1.9911831e+07	10815.0	281	7.37%
30.10	1.81754e+07	4239.0	1.967343e+07	10819.0	289	8.24%
30.11	1.93082e+07	2313.8	2.1254723e+07	10821.0	268	10.08%
30.12	1.83383e+07	3523.5	1.9898886e+07	10820.0	290	8.51%
30.13	1.87637e+07	3742.1	2.0305956e+07	10824.0	271	8.22%
30.14	1.85444e+07	30203.2	1.9933124e+07	10823.0	275	7.49%
30.15	1.81754e+07	4195.4	1.9671333e+07	10820.0	288	8.23%

Na Tabela 5.22 são apresentadas a comparação entre o modelo exato e os melhores resultados obtidos pela heurística BRKGA com relação à média dos resultados para as instâncias com 30 servidores.

Tabela 5.22: Melhores resultados da heurística BRKGA para 30 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
30.1	1.38738e+06	1094.01	1.388062e+06	10802.4	830	0.05%
30.2	1.2334e+06	999.78	1.233897e+06	10803.4	1199	0.04%
30.3	1.2698e+06	927.503	1.270787e+06	10808.6	929	0.08%
30.4	1.23047e+06	884.853	1.232817e+06	10811.9	965	0.19%
30.5	1.5995e+06	1264.48	1.602413e+06	10809.4	817	0.18%

Tabela 5.22 – *Continuação da tabela*

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
30.6	1.93082e+07	2892.4	2.1201849e+07	10829.1	267	9.81%
30.7	1.83383e+07	3545.45	1.9831304e+07	10821.7	296	8.14%
30.8	1.87637e+07	3640.05	2.0242646e+07	10818.9	273	7.88%
30.9	1.85444e+07	13692.5	1.9872337e+07	10800.5	284	7.16%
30.10	1.81754e+07	4239.02	1.9628705e+07	10805.0	290	8.00%
30.11	1.93082e+07	2313.85	2.1161014e+07	10826.3	269	9.60%
30.12	1.83383e+07	3523.47	1.9828124e+07	10828.4	288	8.12%
30.13	1.87637e+07	3742.13	2.0245598e+07	10837.6	269	7.90%
30.14	1.85444e+07	30203.2	1.9865184e+07	10802.2	276	7.12%
30.15	1.81754e+07	4195.41	1.9600152e+07	10825.2	290	7.84%

Pode-se observar na Tabela 5.23 que a distância entre a solução ótima encontrada pelo Modelo Exato e a heurística BRKGA ultrapassou em 5% apenas na instância 50.4, ficando abaixo de 1% para as demais instâncias da Classe A. Mesmo sendo uma classe fácil, a grande quantidade de variáveis, devido ao número de servidores, o código BRKGA utilizou o limite estabelecido de 3 horas.

Tabela 5.23: Validação da heurística BRKGA para 50 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
50.1	2.30765e+06	4018.4	2.320623e+06	10814.0	333	0.56%
50.2	2.14434e+06	2182.6	2.163791e+06	10814.0	395	0.91%
50.3	2.47563e+06	3517.4	2.496218e+06	10811.0	349	0.83%
50.4	2.95794e+06	2991.4	3.149238e+06	10810.0	352	6.47%
50.5	2.29492e+06	3388.7	2.309934e+06	10818.0	301	0.65%

Na Tabela 5.24 são apresentadas a comparação entre o modelo exato e os melhores resultados obtidos pela heurística BRKGA com relação à média dos resultados para as instâncias com 50 servidores.

Tabela 5.24: Melhores resultados da heurística BRKGA para 50 servidores comparados com os resultados do modelo exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
50.1	2.30765e+06	4018.4	2.31914e+06	10811.1	332	0.50%
50.2	2.14434e+06	2182.6	2.161521e+06	10811.2	397	0.80%
50.3	2.47563e+06	3517.3	2.492686e+06	10812.2	350	0.69%
50.4	2.95794e+06	2991.4	3.146987e+06	10820.0	354	6.39%
50.5	2.29492e+06	3388.7	2.308006e+06	10810.5	300	0.57%

Após os resultados obtidos na validação da heurística BRKGA constantes das Tabelas 5.17, 5.19, 5.21 e 5.23, iniciou-se o código para se obter os resultados das instâncias que não foram resolvidas dentro do limite do tempo pelo método exato, instâncias contidas na Tabela 5.5, obtendo-se os seguintes resultados contidos na Tabela 5.25.

Tabela 5.25: Resultados da heurística BRKGA para as Instâncias que não foram resolvidas dentro do tempo de 3 horas pelo Modelo Exatos comparados com os resultados do Modelo Exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
10.17	1.40279e+07	40296.2	1.4374348e+07	10803.0	1469	2.47%
10.19	1.31696e+07	41034.8	1.3432687e+07	10804.0	1482	2.00%
20.16	2.69611e+07	33882.2	2.8068466e+07	10814.0	432	4.11%
20.17	2.63159e+07	29484.6	2.7153495e+07	10817.0	415	3.18%
20.18	2.71112e+07	37732.2	2.8977297e+07	10815.0	438	6.88%
20.19	2.63314e+07	35166.5	2.8353675e+07	10815.0	451	7.68%
20.20	2.65977e+07	30549.4	2.8585148e+07	10813.0	440	7.47%
30.16	1.3639e+12	4806.0	4.2834161e+07	10823.0	214	-100.00%
30.17	1.40965e+12	3789.4	4.2733854e+07	10827.0	225	-100.00%
30.18	1.40304e+12	3632.4	4.1651117e+07	10814.0	223	-100.00%
30.19	1.62524e+12	6183.3	4.3727952e+07	10842.0	196	-100.00%
30.20	1.34715e+12	4926.9	4.3985690e+07	10817.0	208	-100.00%
50.6	4.05993e+07	36072.0	3.3686687e+07	10852.0	100	-17.03%

Tabela 5.25 – *Continuação da tabela*

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempo	Nr Iter	*IND*
50.7	3.64903e+07	12781.2	3.1897567e+07	10854.0	110	-12.59%
50.8	3.28104e+07	2612.3	3.4738681e+07	10854.0	107	5.88%
50.9	4.50675e+07	12738.6	3.6181281e+07	10833.0	107	-19.72%
50.10	4.02088e+07	4102.6	3.3594632e+07	10840.0	113	-16.45%
50.11	4.05059e+07	1974.0	1.8253865e+07	10830.0	227	-54.94%
50.12	3.64903e+07	17294.8	3.18963e+07	10850.0	110	-12.59%
50.13	1.39376e+12	3754.0	3.4724921e+07	10855.0	107	-100.00%
50.14	1.38716e+12	6237.2	3.6189628e+07	10837.0	107	-100.00%
50.15	1.23334e+12	5685.5	3.3582142e+07	10849.0	113	-100.00%
50.16	2.99171e+12	16804.5	2.8458672e+08	10882.0	74	-99.99%
50.17	2.84913e+12	22579.9	6.769961e+07	10878.0	82	-100.00%
50.18	3.2833e+12	15728.0	7.5259736e+07	10866.0	73	-100.00%
50.19	3.13084e+12	1588.6	2.91341011e+08	10884.0	68	-99.99%
50.20	3.1316e+12	1452.0	7.2214136e+07	10871.0	70	-100.00%

Para as instâncias com 10 e 20 servidores, os resultados apresentaram distâncias compatíveis com relação aos testes de validação do código. A grande diferença aparece nas instâncias com 30 e 50 servidores, onde o código BRKGA obteve melhorias significativas, principalmente para as instâncias das Classes C e D com 50 servidores. Acredita-se que esta grande diferença seja por falta de memória para o processamento do código pelo modelo exato, levando a concluir que nestes casos a aplicação de métodos heurísticos são mais vantajosos para obtenção de melhores resultados.

Na Tabela 5.26 são apresentadas a comparação entre o modelo exato e os melhores resultados obtidos pela heurística BRKGA com relação à média dos resultados das instâncias que não foram resolvidas dentro do tempo de 3 horas pelo modelo exato.

Tabela 5.26: Melhores resultados da heurística BRKGA para as Instâncias que não foram resolvidas dentro do tempo de 3 horas pelo Modelo Exatos comparados com os resultados do Modelo Exato.

Serv.Inst	Modelo Exato	Tempo	BRKGA	Tempos	Nr Iter	*IND*
10.17	1.40279e+07	40296.2	1.4359179e+07	10800.4	1462	2.36%
10.19	1.31696e+07	41034.8	1.340922e+07	10802.2	1497	1.82%
20.16	2.69611e+07	33882.2	2.8027971e+07	10820.2	435	3.96%
20.17	2.63159e+07	29484.6	2.713334e+07	10825.1	416	3.11%
20.18	2.71112e+07	37732.2	2.8945084e+07	10816.5	436	6.76%
20.19	2.63314e+07	35166.5	2.8305148e+07	10809.5	445	7.50%
20.20	2.65977e+07	30549.4	2.8520432e+07	10823.7	438	7.23%
30.16	1.3639e+12	4806.05	4.2794284e+07	10817.4	216	-100.00%
30.17	1.40965e+12	3789.35	4.2699779e+07	10845.5	226	-100.00%
30.18	1.40304e+12	3632.45	4.1611054e+07	10816.8	221	-100.00%
30.19	1.62524e+12	6183.29	4.370853e+07	10842.6	196	-100.00%
30.20	1.34715e+12	4926.9	4.3959169e+07	10802.1	208	-100.00%
50.6	4.05993e+07	36072.0	3.3632073e+07	10841.3	100	-17.16%
50.7	3.64903e+07	12781.2	3.1872324e+07	10832.3	110	-12.66%
50.8	3.28104e+07	2612.2	3.4711653e+07	10895.3	108	5.79%
50.9	4.50675e+07	12738.6	3.6139216e+07	10830.5	107	-19.81%
50.10	4.02088e+07	4102.6	3.3525969e+07	10823.2	113	-16.62%
50.11	4.05059e+07	1973.9	3.3664434e+07	10822.3	99	-16.89%
50.12	3.64903e+07	17294.8	3.1872324e+07	10868.0	109	-12.66%
50.13	1.39376e+12	3754.0	3.4675036e+07	10860.3	107	-100.00%
50.14	1.38716e+12	6237.2	3.6158911e+07	10852.5	107	-100.00%
50.15	1.23334e+12	5685.5	3.3537421e+07	10840.8	113	-100.00%
50.16	2.99171e+12	16804.5	2.84559883e+08	10892.9	74	-99.99%
50.17	2.84913e+12	22579.9	6.7671908e+07	10805.9	82	-100.00%
50.18	3.2833e+12	15728.0	7.5233476e+07	10862.5	71	-100.00%
50.19	3.13084e+12	1588.6	2.89881283e+08	10936.3	69	-99.99%
50.20	3.1316e+12	1452.0	7.2192298e+07	10813.9	70	-100.00%

Na Figura 5.1 pode-se observar o comportamento dos algoritmos empregados no trabalho para a instância 30.12.

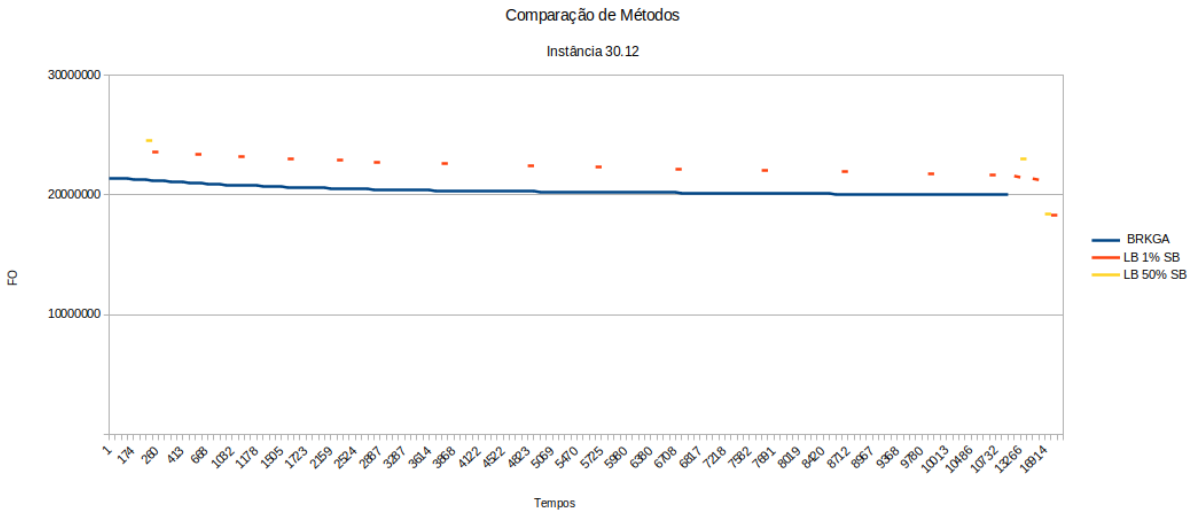

Figura 5.1: Comparação da evolução dos valores da FO para o BRKGA, *Local Branch* com 1% e 50% do Suporte Binário

Na instância 30.12 o BRKGA utilizou 290 iterações e 10820 segundos para se aproximar 8.51% da solução ótima para a instância, sendo que o *Local Branch* com 1% do Suporte Binário utilizou 20 passos para atingir a solução ótima, com um tempo de 21363 segundos, e *Local Branch* com 50% do Suporte Binário utilizou 4 passos para atingir a solução ótima, com um tempo de 42642 segundos.

Na Figura 5.2 pode-se observar o comportamento dos algoritmos empregados no trabalho para a instância 50.14.

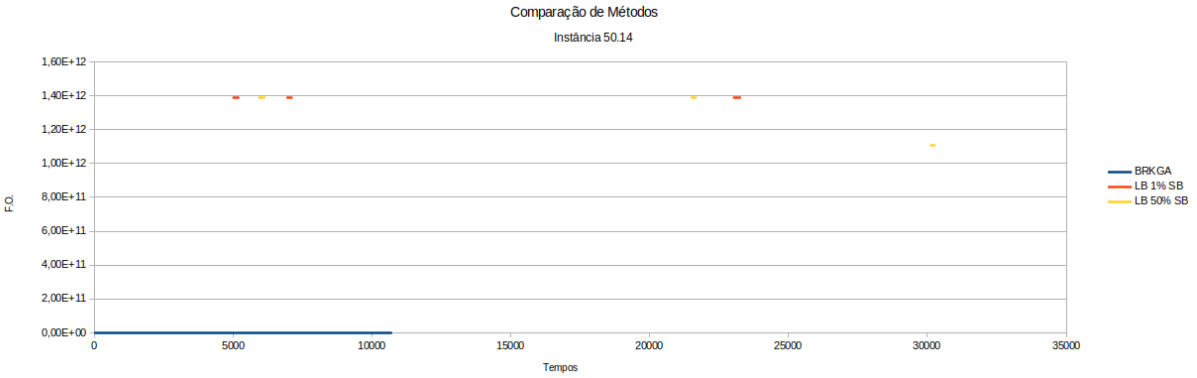

Figura 5.2: Comparação da evolução dos valores da FO para o BRKGA, *Local Branch* com 1% e 50% do Suporte Binário

Na instância 50.14 o *Local Branch* com 1% do Suporte Binário utilizou 03 passos para encontrar um resultado para F.O. de 1.38716e+12, com um tempo de 23287 segundos, o *Local Branch* com 50% do Suporte Binário utilizou 03 passos para encontrar um resultado para F.O. de 1.11023e+12, com um tempo de 30096.0 segundos e o BRKGA conseguiu atingir um valor para a F.O. de 3.6189628e+07, com 107 iterações e com um tempo de 10837.0 segundos, conseguindo obter uma melhora de aproximadamente 100%, mostrando ser mais eficiente nas instâncias que o *Local Branch* não foi possível encontrar a solução dentro do tempo estabelecido para este trabalho.

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

6.1 Conclusões

Com a implementação do algoritmo *Local Branch* podemos concluir que a técnica desenvolvida por Fischetti et al. [25] funciona conforme foi possível observar na validação do código implementado nas Tabelas 5.6, 5.7, 5.8 e 5.9. Pode-se observar nas tabelas de validação do código, o método proposto no presente trabalho encontrou a solução de todas as instâncias resolvidas pelo modelo exato dentro do tempo de 3 horas.

Com relação às instâncias que não foram resolvidas dentro do tempo de 3 horas, o modelo exato obteve melhor desempenho em apenas (03) três instâncias, o algorítmo *local branch* obteve melhor desempenho em (12) doze instâncias e empatou com o modelo exato nas demais instâncias, conforme a Tabela 5.16.

Um dos grandes óbices do algoritmo *Local Branch* aplicado ao PDCRR é a questão do tempo, uma vez que o código pode demorar até 2.8 vezes o tempo de 3 horas.

Cabe enfatizar que a cada passo no *loop* para chamar a função local_branch o algoritmo acrescenta mais duas restrições ao problema, uma para criar a nova vizinhança e outra para excluir a vizinhança que já foi explorada no passo anterior, evitando que seja explorada novamente, ou seja, a cada passo são acrescentadas duas restrições a um problema, aumentando ainda mais as matrizes envolvidas na solução.

O tempo também foi utilizado como limitador do resolvedor CPLEX para encontrar a solução dentro de uma vizinhança e para executar o *loop*, critérios esses que não são previstos por Fischetti et al. [25], em que o critério de parada do método é encontrar

uma solução pior que a solução da vizinhança anterior. O fato de não encontrar a solução ótima dentro de uma vizinhança no tempo determinado e esta vizinhança ser excluída do espaço de busca, pode ocasionar em exclusão da solução ótima do espaço de busca restante, uma vez que a vizinhança não foi totalmente explorada.

Outra questão relevante referente ao PDCRR estudado no presente trabalho é que, mesmo se definindo uma vizinhança pequena através do conjunto binário, a quantidade de variáveis é muito grande. Seria necessário para o algoritmo obter êxito na busca da solução, estabelecer vizinhanças cujo tamanho permitisse ao CPLEX ter condições de encontrar a solução ótima dentro de um tempo aceitável computacionalmente, um equipamento com maior capacidade de memória e o algoritmo não tivesse restrições de tempo para pesquisar as vizinhanças dentro do espaço de busca. Porém, pelos testes realizados, o tempo deste processo foi extremamente elevado, sem contar que o algoritmo poderia estar se aproximando de um ótimo local de baixa qualidade e não da solução ótima do problema, sendo necessário, neste caso, implementar a técnica da diversificação dentro do espaço de busca.

Embora com a implementação do método *local branch* tenha-se obtido bons resultados comparados com o modelo exato, para as instâncias maiores, principalmente as com 50 servidores, não se obteve melhoras significativas tendo em vista a limitação de *hardware*, no tocante a memória, para processar todos os dados das instâncias dentro do tempo de 3 horas. Visando contornar esta limitação e procurar soluções melhores, implementou-se a heurística BRKGA, Algoritmo Genético de Chaves Aleatórias Viciadas.

O BRKGA foi capaz de encontrar boas soluções para resolução do problema onde o modelo exato encontrou a solução ótima dentro do tempo e conseguiu obter melhorias significativas nas instâncias que o modelo exato não conseguiu resolver dentro do tempo computacional estipulado de 3 horas, principalmente nas instâncias maiores e mais complexas.

6.2 Trabalhos Futuros

Como sugestão para trabalho futuro, sugere-se a implementação do algoritmo POP-MUSIC (*Partial OPtimization Meta-heuristic Under Special Intensification Conditions*), Meta-heurística de Otimização Parcial sob Condições Especiais de Intensificação, um método que trabalha com decomposição em problemas menores e projetado para lidar com instâncias grandes de problemas de otimização combinatória difíceis, através de sua de-

composição em problemas menores. A decomposição de um problema em subproblemas pode levar a soluções de qualidade apenas moderada, já que os subproblemas podem ter sido criados de maneira arbitrária. Alternativamente, POPMUSIC parte de uma solução disponível para o problema (não necessariamente boa), e tenta melhorar a qualidade da solução de subconjuntos da solução inicial, a posteriori. A solução inicial é decomposta em "partes"; uma parte é escolhida como "semente", a partir da qual forma um subproblema, de tamanho definido. Cada subproblema é composto por uma coleção das partes mais fortemente relacionadas à semente. Algum método de otimização (exato ou heurístico) é empregado para tentar melhorar localmente o subproblema. O POPMUSIC organiza o processo de montagem e tratamento de cada subproblema, e de transposição das melhorias eventualmente obtidas durante o tratamento de cada subproblema para o problema completo, até que toda a solução tenha sido examinada e não exista mais nenhum subproblema que possa ser melhorado. Se a definição das partes e subproblemas são feitas de maneira adequada, a cada melhora obtida para o subproblema corresponde uma melhora no problema completo.

Além do POPMUSIC, sugere-se o emprego de outras meta-heurísticas mais tradicionais ou meta-heurísticas não populacionais, como *Simulated Annealing* ou VNS, para comparação com o algoritmo genético.

Referências

[1] LABIC. World Wide Web, http://labic.ic.uff.br/, 2005. Acessado em janeiro de 2009.

[2] The Internet in Real Time. World Wide Web, https://betfy.co.uk/internet-realtime/, 2020. Acessado em junho de 2020.

[3] AKAMAI TECHNOLOGIES. World Wide Web, https://www.akamai.com/pt/our-thinking/cdn/, 2021. Acessado em setembro de 2021.

[4] CDN PEERING e PNI - BRASIL. World Wide Web, https://wiki.brasilpeeringforum.org/, 2021. Acessado em setembro de 2021.

[5] CLOUD CDN GOOGLE. World Wide Web, https://cloud.google.com/cdn/docs/overview, 2021. Acessado em setembro de 2021.

[6] Porque precisa de um CDN para WordPress. World Wide Web, https://prnethost.com.br/site/cdn-para-wordpress/, 2021. Acessado em abril de 2021.

[7] Windows Azure - CDN. World Wide Web, https://fabriciosanchez.azurewebsites.net/3/windows-azure-content-delivery-network-cdn/, 2021. Acessado em abril 2021.

[8] Problema de Distribuição de Requisições. Nota de aula do Curso Redes de Computadores, Redes de Computadores II, Aula 04, World Wide Web, https://materialpublic.imd.ufrn.br/curso/disciplina/4/57/4/3, 2022. Acessado em maio de 2022.

[9] Problema de Posicionamento de Réplicas e Distribuição de Requisições. World Wide Web, ttp://www.ic.uff.br/simone/sd/contaulas/aula17.pdf, 2022. Acessado em maio de 2022.

[10] Ahuja, R.K., M. T. O. J. *Network Flows: Theory, Algorithms, and Applications*, 1 ed. Prentice Hall, 1993.

[11] Aioffi, W., Mateus, G., Almeida, J., Loureiro, A. Dynamic content distribution for mobile enterprise networks. *IEEE Journal on Sele ted Areas in Communications 23* (2005), 2022–2031.

[12] Bazaraa, M. S., Jarvis, J. J., (Author, H. D. S. *Linear programming and network flows*. Prentice Hall, 1943.

[13] Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *INFORMS Journal on Computing 6(2)* (1994), 154–160.

[14] Bektas, T., Oguz, O., Ouveysi, I. Designing cost-effective content distribution networks. *Computers and Operations Research 34* (2007), 2436–2449.

[15] Berger, D. S., Sitaraman, R. K., Harchol-Balter, M. *AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network*. Academic Press, 2017.

[16] Bjorkqvist, M., C. L. Y. Z. X. Minimizing retrieval cost of multi-layer content distribution systems. *Communications (ICC), IEEE Internetional Conference* (2011), 1–6.

[17] Boechat, R. G. G. Análise de redundância em formulações matemáticas para um problema de gerenciamento ligado a redes de comunicação do tipo cdn. Dissertação de Mestrado, Universidade Federal Fluminense - UFF, Brasil, 2014.

[18] Boechat, R. G. G. Problema de distribuição de capacidades, réplicas e requisições. Dissertação de Mestrado, Universidade Federal Fluminense - UFF, Brasil, 2017.

[19] Boechat, R. G. G., Neves, T. A., Albuquerque, C. Análise de redundância em uma formulação matemática para o problema de posicionamento de réplicas e distribuição de requisições em redes de distribuição de conteúdos. *SBPO - Simpósio Brasileiro de Pesquisa Operacional 8* (2015).

[20] Chen, F., Guo, K., Lin, J., Porta, T. L. Intra- loud lightning: Building cdns in the cloud. *IEEE Infocom* (2012), 433–441.

[21] Darwin, C. A. *On the origin of species*. New York :D. Appleton and Co., 1859.

[22] de Paula Junior, U. C., Drummond, L. M. A., Frota, Y., Simonetti, L. Posicionamento de réplicas em redes de distribuição de conteúdos. Em *XLIII SBPO* (2011).

[23] Eiben, A. E., Smith, J. E. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[24] Filippetti, M. A. *CCNA 4.1 - Guia Completo de Estudo*. Visual Books, 2008.

[25] Fischetti, M., Lodi, A. Local branching. *Mathematical Programming Series B 98* (2003), 23–47.

[26] Garg, A., Gupta, M. Improving qos by enhancing media streaming algorithm in content delivery network. *International Journal of Engineering and Advanced Technology 8* (2019), 866–870.

[27] Garg, A., Gupta, M. A perusal of replication in content delivery network. *LOBIYAL, D.; MANSOTRA, V.; SINGH, U. (eds) Next-Generation Networks. Advances in Intelligent Systems and Computing 638* (Springer Singapore, 2018).

[28] Gendreau, M., Potvin, J.-Y. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2 a edição, 2010.

[29] Goldbach, R. Popmusic aplicada ao problema de posicionamento de réplicas e de distribuição de requisições em redes de distribuição de conteúdos. Dissertação de Mestrado, Universidade Federal do Estado do Rio de Janeiro - UERJ, Brasil, 2013.

[30] Gonçalves, J. F., Resende, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics 17(5)* (2010), 487–525.

[31] Hansen, P., Mladenovi¸, N. Pesquisa de vizinhança variável. *Computers and Operations Research 24* (1997), 1097–1100.

[32] Hansen, P., Mladenovi¸, N. Variable neighborhood search. *Glover F, Kochenberger G, editors. Hand book of metaheuristics. Boston, Dordrecht, Londres: Kluwer Academic Publisher* (2003), 145–184.

[33] Hansen, P., Mladenovi¸, N., Urošević, D. Variable neighborhood search and local branching.

[34] Hu, H., Wen, Y. Joint content replication and request routing for social video distribution over cloud cdn: A community clustering method. *IEEE Transactions on Circuits and Systems for Video Technology 26* (2016), 1320–1333.

[35] Igder, S., Bhattacharya, S., Qazi, B. R., Elmirghani, J. M. H. Standalone green cache points for vehicular content distribution networks. *10th International Conference on Next Generation Mobile Applications, Security and Technologies* (2016), 59–64.

[36] Lalla-Ruiz, E., Voss, S., Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, J. M. A popmusic-based approach for the berth allocation problem under time-dependent limitations. Em *Annals of Operations Research* (2015), p. 871–897.

[37] Laoutaris, N., Zissimopoulos, V., Stavrakakis, I. On the optimization of storage capacity allocation for content distribution. *Computer Networks 47* (2005), 409–429.

[38] Li, F., Xu, L., Duan, S., Wu, W., Zhao, H., Ling, Q. Improving hierarchical mobile video caching through distributed cross-layer coordination. Em *Multimedia Tools and Applications* (2019), p. 6049–6071.

[39] Li, W., Chan, E., Wang, Y., Chen, D., Sanglu. Cache placement optimization in hierarchical networks: Analysis and performance evaluation. *International Conference on Research in Networking* (2008), 385–396.

[40] Lin, C. F., Leu, M. C., Chang, C. W., Yuan, S. M. The study and methods for cloud based cdn. *Cyber-Enabled Distributed Computing and Knowledge Discovery (Cyber C* (2011), 469–475.

[41] Maniezzo, V., Stützle, T., Voÿ, S., Eds. *Metaheuristics*, 1 ed. Springer, 2009.

[42] Neves, T., Drummond, L., Ochi, L., Albuquerque, C., Uchoa, E. Solving replica placement and request distribution in content distribution networks. *Electronic Notes in Discrete Mathematics 36* (August 2010), 89–96.

[43] Neves, T., Ochi, L., Albuquerque, C. A new hybrid heuristic for replica placement and request distribution in content distribution networks. Em *Optimization Letters* (2015), p. 677–692.

[44] Neves, T. A. Redes de distribuição de conteúdos: Abordagens exatas e heurísticas. Relatório Técnico, UFF, 2011.

[45] Rochman, A. N., Prasetyo, H., Nugroho, M. T. Biased random key genetic algorithm with insertion and gender selection for capacitated vehicle routing problem with time windows (2017). 1–5.

[46] Silva, R., Resende, M., Pardalos, P., Gonçalves, J. Biased random-key genetic algo- rithm for bound-constrained global optimization. Em *Global Optimization Workshop* (2012), p. 133–136.

[47] Song, Z., Berger, D. S., Li, K., Lloyd, W. Learning relaxed belady for content distribution network caching. *17th USENIX Symposium on Networked Systems Design and Implementation* (2020), 529–544.

[48] Spears, V. M., Jong, K. A. D. On the virtues of parameterized uniform crossover. Em *Fourth International Conference on Genetic Algorithms* (1991), p. 230–236.

[49] Srivastar, M. K., Bhadoria, R. S., Pramanik, T. Integration of multiple cache server scheme for user-based fuzz logic in content delivery networks. *Handbook of Research on Advanced Applications of Graph Theory in Modern Society, IGI Global* (2020), 386–396.

[50] Taillard, E. D., Voss, S. Popmusic — partial optimization metaheuristic under special intensification conditions. Em *Essays and Surveys in Metaheuristics. Operations Research/Computer Science Interfaces Series* (2012).

[51] Tang, X., Xu, J. On replica placement for qos-aware content distribution. Em *Proc. of INFOCON2004* (2004), p. 806–815.

[52] Toso, R. F., Resende, M. G. C. A c++ application programming interface for biased random-key genetic algorithms. optimization methods and software. Relatório Técnico, 2014.

[53] Uderman, F., Neves, T., Albuquerque1, C. Optimizing server storage capacity on content distribution networks. *XVI Workshop de Gerência e Operação de Redes e Serviços* (2011), 145–184.

[54] Venketesh, P., Venkatesan, R. A survey on applications of neural networks and evolution- arytechniques in web caching. *IETE Technical Review, 26* (2009), 171–180.

[55] WEN, Y., CHEN, Y., SHAO, M.-L., GUO, J.-L., LIU, J. An efficient content distribution network architecture using heterogeneous channels. *IEEE - SPECIAL SECTION ON DATA MINING INTERNET OF THINGS 8* (2020), 210988–211006.

[56] Wu, J., Ravindran, K. Optimization algorithms for proxy server placement in content distribution. Em *Integrated Network Management-Workshops, 2009, IM'09. IFIP/IEEE International Symposium* (2009), p. 193–198.

[57] Wu, Y., Wu, C., Li, B., Qiu, X., Lau, F. Cloud media: When cloud on demand meets video on demand. Em *Conf. Distrib. Comput. Syst. (ICDCS)* (2011), p. 268–277.

[58] Xu, K., Li, X., Bose, S. K., Shen, G. Joint replica server placement, content caching, and request load assignment in content delivery networks. Em *IEEE Access* (2018), p. 17968–17981.

[59] Zhou, X., Xu, C.-Z. Efficient algorithms of video replication and placement on cluster of streaming servers. *Journal of Network and Computer Applications 30* (2007), 510–540.