Universidade Federal Fluminense

TIAGO ARAÚJO NEVES

Redes de Distribuição de Conteúdos: Abordagens Exatas e Heurísticas

NITERÓI

2011

TIAGO ARAÚJO NEVES

Redes de Distribuição de Conteúdos: Abordagens Exatas e Heurísticas

Orientador:

Luiz Satoru Ochi

Coorientador:

Célio Neves Albuquerque

Universidade Federal Fluminense

NITERÓI

2011

Redes de Distribuição de Conteúdos: Abordagens Exatas e Heurísticas

Tiago Araújo Neves

Relatório Técnico complementar da Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor em Computação.

Niterói,18 de Maio de 2011.

Para meus pais, minha irmã e meus avós

Agradecimentos

A Deus, pela saúde e pela ajuda que mais ninguém pode dar.

À minha família, em especial à minha avó Tereza, aos meus pais Willes e Consolação, à minha irmã e ao meu tio Afonso pelo apoio e incentivo.

À Vanda pelo apoio ao longo de todo o doutorado.

Aos todos os professores, e em especial para os professores Luiz Satoru Ochi e Célio Albuquerque pelo apoio e orientação.

Ao amigo Marcone Jamilson Freitas Souza, pelo companheirismo e amizade.

Aos bons amigos que fiz aqui em Niterói, em especial ao Jaques, Puca, Anand, Diego e Luciene.

Aos irmãos de república e laboratório.

Aos eternos amigos de hoje e sempre que estenderam a mão para me apoiar sempre que precisei.

A todos os demais que contribuíram para a realização deste trabalho.

Resumo

Uma Rede de Distribuição de Conteúdos (RDC) é um sistema de computadores interligados através da internet que colaboram para fornecer conteúdos para clientes. Ela é uma rede sobreposta que mantém réplicas de cada conteúdo em seus servidores, com o objetivo de reduzir os atrasos, a carga nos servidores e o congestionamento da rede, melhorando a qualidade do serviço fornecido. Em arquiteturas de RDC tradicionais, as requisições são recebidas por um servidor central e redirecionadas para o servidor mais próximo do cliente. Entretanto, devido aos custos de manutenção das réplicas, não é razoável replicar os conteúdos em toda a rede. Neste trabalho são apresentadas abordagens exatas e heurísticas para resolver um problema de gerenciamento das RDC, conhecido na literatura como Problema de Posicionamento de Réplicas. Este problema consiste em decidir quais réplicas serão colocadas em cada servidor. O objetivo final é minimizar o tráfego na rede sem violar as restrições de qualidade de serviço.

Abstract

A content distribution network (CDN) is a system of computers networked together across the Internet that cooperate to deliver content to clients. It is an overlay network that maintains replicas of each content in its servers, with the goal of reducing delays, server load and network congestion, therefore improving the quality of the service provided. In traditional CDN architectures, requests are received by a central server and then redirected to the server that is closer to the client. However, because of the costs involved in the maintenance of such replicas, it is not reasonable to replicate the contents over the entire network. In this work, exact and heuristic approaches are presented to solve a problem that appears in CDN management, known in the literature as Replica Placement Problem. This problem consists of deciding which replicas will be placed in each server. The overall objective is to minimize the traffic in the network without violating QoS constraints.

Palavras-chave

1. Redes de Distribuição de Conteúdos
2. Formulação Matemática
3. Heurística Construtiva
4. Problema de Posicionamento de Réplicas
5. Qualidade de Serviço
6. Método Híbrido

Glossário

DNS	:	*Domain Naming Service*
IC	:	Instituto de Computação
ILS	:	Iterated Local Search
PPR	:	Problema de Posicionamento de Réplicas
PDR	:	Problema de Distribuição de Réplicas
PPRDR	:	Problema de Posicionamento de Réplicas e Distribuição de Requisições
PLFC	:	Problema de Localização de Facilidades Capacitado
P2P	:	Peer-to-Peer
QoS	:	*Quality of Service*
RDC	:	Rede de Distribuição de conteúdos
RTR	:	Record-To-Record
RTT	:	*Round Trip Time*
UFF	:	Universidade Federal Fluminense
URL	:	*Uniform Resource Locator*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o crescimento da internet, a demanda por conteúdos diversos aumentou consideravelmente em todo o mundo. Alguns destes conteúdos, em especial os conteúdos de multimídia, necessitam de algum tipo de suporte especializado, que possa oferecer garantias de Qualidade de Serviço (Quality of Service - QoS) como atraso máximo e largura de banda mínima para que as expectativas dos clientes possam ser satisfeitas. Neste sentido, muitos estudos têm sido feitos na tentativa de encontrar maneiras de servir melhor a crescente demanda por conteúdos com restrições de QoS [46, 52].

Um modo de servir os conteúdos multimídia de maneira eficiente é através do uso de Redes de Distribuição de Conteúdos (RDC) [12, 46, 52], que são tipicamente redes sobrepostas usadas para posicionar réplicas dos conteúdos nas proximidades dos clientes, reduzindo assim, o atraso, a carga nos servidores e o congestionamento da rede, possibilitando portanto, melhorias na qualidade do serviço prestado.

Em arquiteturas de RDC tradicionais, as requisições dos clientes são recebidas por um servidor central que as redireciona para outros servidores capazes de atender a requisição e que se encontrem nas proximidades do cliente. Na realidade, as requisições dos clientes nem sempre serão atendidas pelo servidor mais próximo. Em muitos casos, é mais vantajoso usar um servidor que se encontra um pouco mais distante, mas que esteja menos sobrecarregado.

Empresas como Akamai [1], Level 3 [2] e Mirror Image [3] especializaram-se em prover arquiteturas de RDC para empresas, como Adobe, Aude Ag e Fox Interactive, que querem, ou necessitam distribuir seus conteúdos.

Existem vários problemas de otimização dentro das arquiteturas de RDC. Entre eles estão o Problema de Localização de Servidores (PLS), o Problema de Replicação (PR) e

o Problema de Posicionamento de Réplicas (PPR). O PLS pertence à classe NP-Difícil e consiste em encontrar os melhores locais dentro da rede real para posicionar os servidores que irão compor a rede sobreposta da RDC de modo a minimizar o tráfego [14]. O PR pertence à classe P e consiste em decidir quais conteúdos serão replicados [52]. E o PPR pertence à classe de problemas NP-difíceis e consiste em encontrar os melhores locais (servidores) dentro da rede sobreposta para colocar os conteúdos replicados de modo a reduzir os custos de transmissão [8, 52].

O problema que é abordado neste trabalho, chamado Problema de Posicionamento de Réplicas e Distribuição de Requisições ou PPRDR, é uma variante do PPR e consiste em encontrar os melhores locais para posicionar as réplicas dos conteúdos a fim de que as exigências de QoS das requisições não sejam violadas e o tráfego dentro da rede seja minimizado.

Do conhecimento dos autores, não existe nenhum trabalho para o PPRDR como enfocado neste trabalho, no entanto, existem vários trabalhos na literatura que tratam de problemas semelhantes ao PPRDR usando abordagens distintas, tais como o proposto por [52] que trata o PR e o PPR de forma isolada e o apresentado por [8] que trata os mesmos problemas de maneira conjunta. Em relação ao PPR, a questão da distribuição de requisições é abordada na literatura de duas formas, ou através do uso do algoritmo de Dijkstra [49] ou através de uma análise de custos [14] [46].

Este trabalho trata de maneira conjunta o PR, o PPR e a distribuição de requisições para arquivos extensos, analisando não apenas os custos para associar as requisições aos servidores, mas também uma série de questões que até então não haviam sido abordadas de maneira conjunta, como banda mínima para os clientes, carga nos servidores e presença de múltiplos conteúdos. Além dessas questões, uma série de outras, relacionadas ao fato dos conteúdos serem extensos, como possibilidade de atendimento por mais de um servidor ao mesmo tempo e a possibilidade de um atendimento se estender por vários períodos de tempo, também são consideradas.

O PPRDR é um problema de grande aplicação prática e para tratá-lo três abordagens distintas são propostas: uma abordagem exata para o modelo offline, através do uso de formulações matemáticas, e abordagens heurísticas e híbridas para o modelo online. O objetivo do uso de três abordagens distintas é realizar um estudo mais aprofundado a respeito da melhor forma para tratar este problema real, que é tipicamente encontrado em ambientes de RDC.

Dentre as contribuições deste trabalho estão: 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. Um *framework* unificador para heurísticas construtivas para a versão *online* do PPRDR tratado pela formulação *FD*. Abordagens exatas e heurísticas para o PPR. Abordagens exatas e heurísticas para o PDR. Análise da complexidade do PDR, que prova que este problema pertence à classe P. Abordagens híbridas compostas como combinação de abordagens para o PPR e para o PDR. Versões otimizadas (através da inclusão de componentes exatos) de heurísticas utilizadas em mercado e apresentadas pela literatura.

O restante deste trabalho está organizado da seguinte maneira: no Capítulo 2 é feita uma breve exposição sobre o funcionamento de uma RDC. O Capítulo 3 faz uma descrição detalhada do PPRDR. O Capítulo 4 apresenta uma revisão bibliográfica a respeito do tema. O Capítulo 5 expõe uma discussão a respeito de algumas abordagens exatas para o problema. Já no Capítulo 6 são apresentadas as abordagens heurísticas centralizadas propostas para o problema. O Capítulo 7 mostra como foram geradas as instâncias para o problema bem como suas principais características. O Capítulo 8 expõe os resultados obtidos e no Capítulo 9 encontram-se as conclusões parciais do trabalho e algumas propostas para trabalhos futuros.

1.1 Motivação

O PPRDR, como enfocado nesta tese, considerando de maneira conjunta a replicação, o posicionamento das réplicas e distribuição eficiente das requisições, ainda é pouco explorado pela literatura apesar de sua visível importância econômica, fato este que justifica e motiva o estudo de maneiras eficientes para atacar este problema. Além disso, outra grande motivação deste trabalho é a possibilidade de melhorar a qualidade e a eficiência dos serviços prestados pelos provedores de RDC. Estas duas motivações vão de encontro à demanda mundial por melhor aproveitamento de recursos vivenciada neste século, onde simplesmente apresentar soluções para os problemas não é mais suficiente. É necessário apresentar soluções eficientes, que satisfaçam as necessidades dos cliente e que aproveitem de maneira racional os recursos disponíveis.

Capítulo 2

Redes de Distribuição de Conteúdos

Uma Rede de Distribuição de Conteúdos (RDC) é um sistema de computadores interligados através da internet que colaboram para fornecer conteúdos para clientes. Ela é uma rede sobreposta de alto desempenho que mantém réplicas de cada conteúdo em seus servidores, com o objetivo de tentar reduzir: os atrasos, a carga nos servidores e o congestionamento da rede, melhorando assim, a qualidade do serviço.

Em um sistema cliente-servidor de Internet típico [29], um cliente estabelece uma conexão com um servidor tipicamente através de um navegador, que é uma ferramenta amplamente utilizada para o acesso aos servidores da rede. Um servidor, por sua vez, disponibiliza arquivos no formato de hipertexto e objetos, que, quando agrupados, são chamados de *sites*. Na internet, um caminho para um servidor é identificado por uma URL (do inglês *Uniform Resource Locator*).

Uma página da internet é composta de um documento base, escrito em alguma linguagem de marcação, HTML, por exemplo, e uma série de objetos incorporados, como imagens, arquivos de vídeo e áudio. Devido a isso, páginas com vinte ou mais objetos são extremamente comuns. Cada uma das imagens e vídeos é um objeto independente e é entregue separadamente. O comportamento comum de um cliente de internet é baixar primeiro o documento base e, imediatamente depois, os objetos incorporados que estão, em geral, no mesmo servidor.

A Figura 2.1 mostra como é a estrutura de um sistema cliente servidor típico no paradigma da internet. Nela são mostrados três clientes (C1,C2 e C3) requisitando dois conteúdos diferentes para um único servidor central. As requisições são indicadas pela letra *Q* seguida de um número, assim, a sequência *Q1* representa uma requisição para o conteúdo *1*, e a sequência *Q2* representa requisições para o conteúdo *2*.

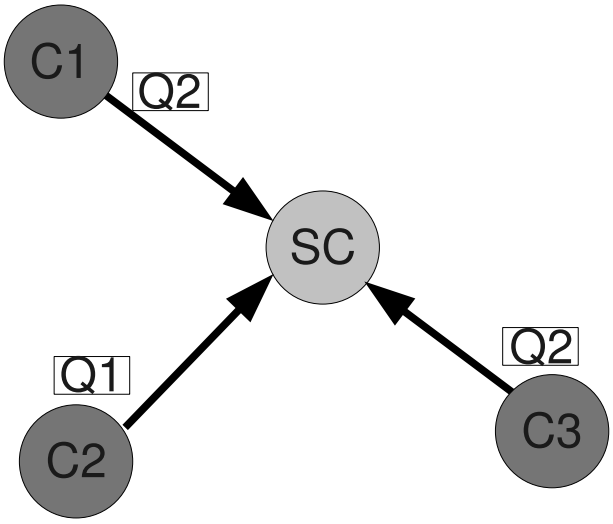

Figura 2.1: Sistema Cliente-Servidor típico

Uma RDC tipicamente age como intermediária entre os clientes e os servidores. Em geral, os objetos incorporados a um *site* estão localizados no mesmo servidor que o documento base, mas isso não é realmente uma necessidade. Os objetos incorporados também são descritos através de URLs, o que possibilita que eles estejam localizados em um servidor diferente do documento base.

Assim, a RDC replica os objetos incorporados, também chamados de conteúdos, e posiciona estrategicamente estas réplicas em servidores próximos aos clientes, procurando com isso reduzir o atraso percebido por estes bem como o tráfego total na rede. Como consequência lógica da existência de múltiplas réplicas de um conteúdo posicionadas em servidores diferentes, obtém-se a redução da carga de trabalho nos servidores uma vez que existe a possibilidade de que cada servidor atenda somente a uma fração do total de demandas.

A Figura 2.2 mostra um possível uso de uma arquitetura de RDC para o cenário proposto na Figura 2.1. Na Figura 2.2 são mostrados os mesmos clientes, com as respectivas requisições, da Figura 2.1, porém, o papel do servidor central é desempenhado por uma RDC de dois servidores (*S1* e *S2*). Estes servidores abrigam réplicas dos conteúdos, representadas pela letra R seguidas por um número que representa qual conteúdo está replicado. Assim, a sequência *R1* representa uma réplica do conteúdo *1*.

Ao se comparar as duas figuras, considerando que no caso ilustrado a distância euclidiana entre os clientes e os servidores aos quais os clientes estão ligados nas figuras seja proporcional ao tempo gasto para transmissão dos conteúdos, constata-se que o tempo de

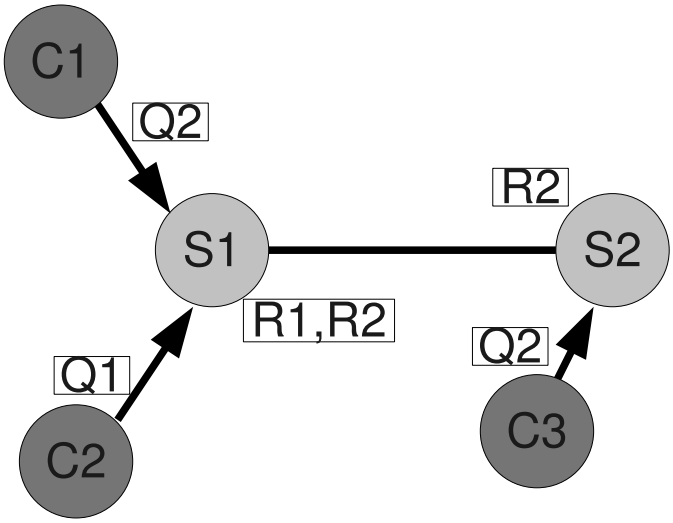

Figura 2.2: Sistema Cliente-Servidor com RDC

atendimento de cada um dos clientes mostrados na Figura 2.2 tende a ser menor do que aqueles apresentados pelo cenário exposto na Figura 2.1. A carga de trabalho nos servidores também é reduzida com o uso de uma RDC, visto que, esta carga é dividida entre os servidores da RDC e não atendida em sua totalidade por um único servidor central, como ilustra a Figura 2.1.

Para disponibilizar o conteúdo em uma RDC, depois de estabelecidas as questões contratuais, o contratante deve modificar o documento base do conteúdo. Preferencialmente, o documento base deve ser provido pelo *site* do contratante e os objetos incorporados por outros servidores, sendo o ideal que estes servidores estejam posicionados nas proximidades do cliente, que não estejam sobrecarregados e que já possuam uma cópia do objeto requisitado.

Para isso, a URL de cada um dos objetos que serão distribuídos pela RDC 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 de RDC. 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 RDC. 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 RDC, e não para um servidor controlado pelo fornecedor do conteúdo. Entre os parâmetros que podem ser considerados estão a data de criação do objeto, seu tamanho, sua popularidade antecipada e seu tipo (foto, vídeo, etc.) e o conteúdo em si. Com estas informações, é possível fazer balanceamento de carga, direcionamento de fluxo de um determinado tipo

para um conjunto específico de servidores e ainda evitar que o cliente receba conteúdo desatualizado. 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 estrutura da RDC pode encontrar outro servidor virtual utilizando os parâmetros da função de *hash*.

A Figura 2.3 mostra como uma requisição de um cliente é processada em uma arquitetura de RDC.

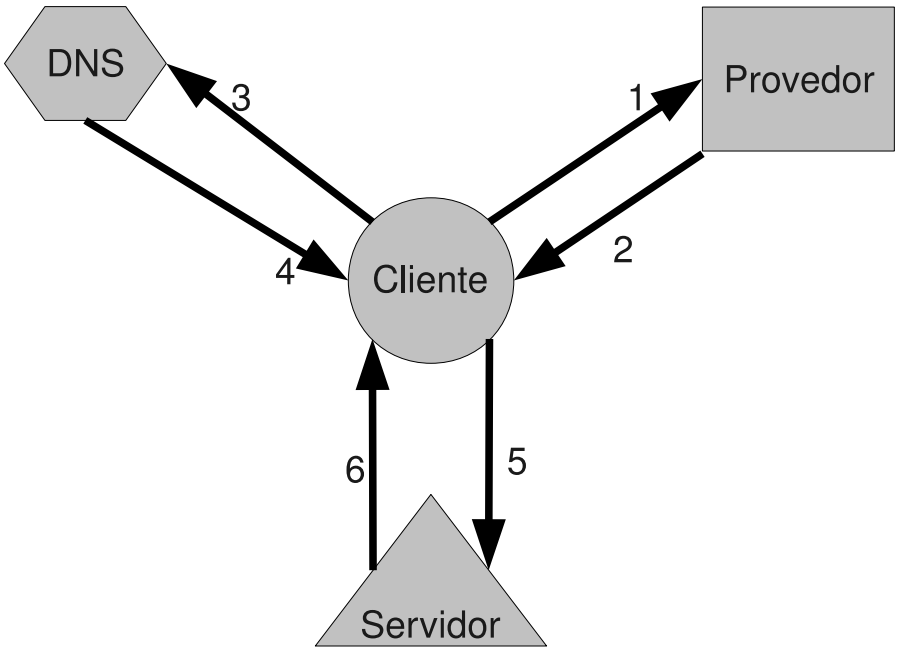

Figura 2.3: Processamento de uma requisição em uma RDC

Primeiramente, como mostrado no passo 1 da Figura 2.3, o cliente envia uma requisição por um determinado conteúdo para o site do provedor, como se o conteúdo não estivesse replicado pela RDC. A diferença é que, ao invés da página usual, o provedor vai enviar para o cliente a página com as URLs modificadas, como mostrado no passo 2. Manter o documento base no provedor possibilita que o conteúdo seja atualizado sem muitas dificuldades. Quando uma página é modificada, a função de *hash* para cada um dos objetos incorporados deve recalculada e as URLs reescritas. Como o provedor de conteúdos é que define quais objetos inserir em sua página, é natural que ele faça as devidas alterações e depois use ferramentas adequadas para que a página seja disponibilizada em uma RDC.

Em uma segunda fase, correspondente aos passos 3 e 4 da Figura 2.3, a requisição é tratada por uma cadeia de servidores DNS que, entre outras coisas, analisa a origem da requisição para determinar um servidor próximo ao cliente e que não esteja sobrecarregado para tratar esta requisição. Há dois possíveis casos nesta fase: requisições para conteúdos pequenos e requisições para conteúdos grandes. Para conteúdos pequenos, a cadeia de DNS irá associar a requisição ao servidor mais próximo da origem da requisição e que, preferencialmente, já possua uma réplica do conteúdo desejado. Se ele possuir a réplica, a requisição é atendida prontamente, caso contrário, uma cópia do conteúdo é solicitada para o provedor deste conteúdo ou para algum outro servidor da RDC que possua sua réplica, armazenada para acessos futuros, e enviada ao cliente. Para arquivos grandes, tipicamente arquivos multimídia, que devido ao seu tamanho não podem ficar transitando pela rede, as requisições são direcionadas para servidores específicos, adaptados para este tipo de conteúdos.

Por fim, após determinado o servidor, o conteúdo é requisitado e enviado ao cliente, correspondendo aos passos 5 e 6 da Figura 2.3. Para arquivos grandes, como vídeos e áudios, um processo de análise de desempenho é feito durante a transmissão para verificar se é possível melhorar o serviço prestado através de uma mudança de servidor. Estas mudanças podem ocorrer com frequência, já que, no ambiente de internet, mudanças nos canais de comunicação são bastante comuns. Para exemplificar uma situação de mudança de servidor para prover uma melhora na qualidade do serviço, suponha que existam dois servidores para atender um cliente e que o servidor mais próximo ao cliente esteja sobrecarregado devido à transmissão de outros conteúdos. Em um primeiro momento, devido à sobrecarga do servidor mais próximo, o cliente é atendido pelo servidor que se encontra mais distante. Com o passar do tempo, o servidor que se encontra mais próximo ao cliente termina a transmissão dos outros conteúdos tornando-se apto a atender o cliente. Ocorre neste instante uma mudança no atendimento, e o cliente passa a ser atendido pelo servidor que se encontra mais próximo a ele. Este tipo de mudança de servidor durante o processo de atendimento das requisições possibilita um melhor uso da estrutura da RDC, pois provê uma melhor qualidade aos clientes e reduz o tráfego na rede.

Capítulo 3

O Problema de Posicionamento de Réplicas e Distribuição de Requisições

O Problema de Posicionamento de Réplicas (PPR) consiste em encontrar os melhores servidores para armazenar as réplicas dentro da RDC de modo que toda a demanda seja atendida. Portanto, para resolver este problema, é necessário conhecer as posições dos servidores, bem como sua capacidade de banda e armazenamento.

Como mencionado anteriormente no Capítulo 1, encontrar o posicionamento ótimo para os servidores também pode ser caracterizado como um problema de otimização. Por simplificação, neste trabalho, considerou-se que este problema já está resolvido, e a solução deste é dada como entrada para o problema tratado.

O problema tratado neste trabalho é uma variante do PPR chamada Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR), o qual consiste em encontrar o melhor posicionamento para as réplicas dentro da rede sobreposta e redistribuir as requisições entre os servidores, com o objetivo de reduzir a carga da rede sem violar as restrições de QoS das requisições. Este problema difere PPR pelo fato de que as requisições são tratadas individualmente. Assim, requisições originadas de um mesmo ponto para um mesmo conteúdo podem ser tratadas por servidores diferentes caso isso seja vantajoso.

Requisições com 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. Neste trabalho, os cenários foram construídos considerando a possibilidade da existência de tratamento diferenciado de clientes pela arquitetura da RDC. Portanto, o uso de restrições de QoS surge como opção natural para modelar este tratamento diferenciado, sendo neste trabalho consideradas duas exigências de QoS:

banda mínima e atraso máximo.

A redistribuição de requisições é usada como mecanismo de balanceamento de carga na rede. O objetivo é reduzir a carga nos servidores e melhorar a qualidade do serviço. Em um sistema de redistribuição simples, as requisições são atendidas pelos servidores geograficamente mais próximos, mas, em alguns casos, pode ser vantajoso, com o objetivo de melhorar a qualidade percebida pelo cliente, usar um servidor que se encontra um pouco mais distante mas que não esteja tão carregado quanto o servidor mais próximo.

Para fazer a redistribuição de requisições no PPRDR alguns fatores precisam ser considerados. Uma requisição pode ser redirecionada para um servidor somente se este servidor possui uma réplica do conteúdo desejado pela requisição. As restrições de QoS da requisição também precisam ser averiguadas. Se as restrições são violadas, redirecionar esta requisição irá produzir uma solução inviável. Em alguns casos, para otimizar a distribuição de requisições, é necessário, primeiramente, alterar o posicionamento das réplicas.

O posicionamento das réplicas está sujeito a fatores relacionados com os servidores como, por exemplo, capacidade de armazenamento e distâncias entre os mesmos. No início, os conteúdos se encontram em seus servidores de origem, podendo um ou mais conteúdos ter o mesmo servidor de origem, e só depois, elas são distribuídas pela rede. Devido à possibilidade da existência de custos associados com o 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 e/ou inviabilidades nos requisitos de QoS.

Informações sobre a rede sobreposta também se fazem necessárias. Como posicionar os servidores não é o escopo deste trabalho, considerou-se que estas informações são dadas como entrada para este problema. Assim, informações como distância entre servidores e o atraso na rede são simplesmente dados de entrada neste trabalho.

A Figura 3.1 mostra um exemplo do PPRDR em uma RDC. A figura mostra dois servidores e três clientes. O servidor 1 possui uma réplica do conteúdo 1 ($R1$) e uma do conteúdo 2, o servidor 2 possui uma réplica do conteúdo 2 , e cada cliente envia uma requisição (Q) para um conteúdo na RDC.

Descrito assim, o PPRDR pode ser visto como uma variante do Problema de Localização de Facilidades capacitado com multiproduto e demanda divisível [27, 41], que

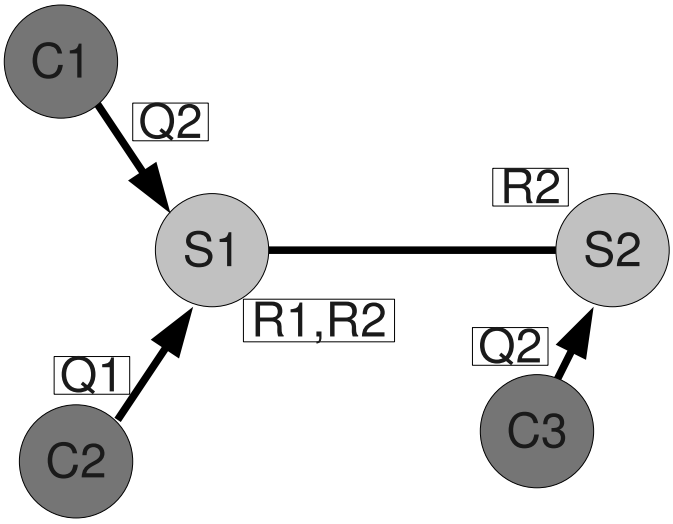

Figura 3.1: O Problema de Posicionamento de Réplicas e Distribuição de Requisições

é uma variante do problema de localização de Facilidades Capacitado (PLFC) [27] que pode ser descrito da seguinte maneira: dado um conjunto C de clientes, um conjunto F de possíveis localizações para abrir facilidades, deve-se determinar um subconjunto de F chamado $F_{abertos}$ de tamanho menor ou igual a um parâmetro k de tal maneira que o custo de atendimento dos clientes em C pelas facilidades em $F_{abertos}$ seja minimizado.

Estabelecendo uma relação entre servidores e facilidades, uma relação entre clientes e requisições e uma outra relação entre produtos e conteúdos, o PLFC e o PPRDR tornam-se similares. Com isso, o uso de técnicas já consagradas para o problema de localização de facilidades torna-se possível para o PPRDR o que até então, segundo o conhecimento dos autores, não havia sido feito. Além disso, se apenas um conteúdo for considerado, o PPRDR pode ser reduzido ao PLFC onde há abertura fechamento e reabertura de facilidades descrito em [19]. Isto faz do PPRDR tratado nesta tese uma generalização do PLFC apresentado em [19] que pertence à classe NP-difícil.

A demanda divisível do problema é uma característica que indica que um cliente (requisição) pode ser atendido por mais de uma facilidade (servidor). É comum encontramos, na literatura, referências sobre o problema da demanda divisível [19, 21, 27]. Isto possibilita o melhor uso dos recursos, utilizando ao máximo as facilidades que possuem um menor custo de entrega de produtos, reduzindo o custo total. Por exemplo, supondo que haja um cliente com demanda de 15 unidades por um produto e duas facilidades para o atendimento, uma com 10 unidades disponíveis e custo de atendimento 1 para cada unidade de demanda e a outra com 20 unidades disponíveis com custo de atendimento 3 por unidade de demanda. Se não permitirmos a divisibilidade, a escolha será a segunda

facilidade totalizando um custo 45. Se a divisibilidade for possível, a escolha será atender este cliente com 10 unidades vindas da primeira facilidade e 5 da segunda, totalizando um custo de 25. Neste cenário, fica claro que a divisibilidade das demandas possibilita a redução de custos ao se lidar com conteúdos extensos. Além disto, esta divisibilidade possibilita o uso de políticas de balanceamento de carga e proporciona uma maior tolerância a falhas no sistema como um todo. Assim sendo, conclui-se que a divisibilidade é uma das características fundamentais para a infraestrutura de uma RDC.

A restrição de capacidade indica que as facilidades possuem uma capacidade limitada e o multiproduto indica que mais de um produto (conteúdo) são considerados ao mesmo tempo.

Considerando o problema de localização de facilidades clássico, o mais comum é encontrar problemas onde a demanda é inteira, afinal, na maioria absoluta dos casos, os produtos não podem ser divididos em frações sem que isso comprometa sua qualidade. Por exemplo, não faz sentido entregarmos um quarto de um pão francês, ou meio tijolo. Contudo, em alguns casos, principalmente quando tratamos de medidas que representam conjuntos, as demandas podem assumir valores fracionários, por exemplo, quando lidamos com quilos de pão ou milheiros de tijolos. Para o PPRDR, como cada produto é um conteúdo diferente, podemos dizer que a demanda é inteira se pensarmos que cada requisição exige um conteúdo. Porém, se considerarmos que cada conteúdo é na verdade um conjunto de Kilobytes ou Megabytes armazenados nos servidores, podemos ter a possibilidade de que a demanda assuma valores fracionários. O uso de uma demanda fracionária torna o problema menos complexo, pois reduz o número de variáveis inteiras necessárias à sua resolução e também facilita na questão da divisibilidade de uma demanda entre vários servidores.

Tratando do PPRDR dinâmico, no qual os dados sofrem alterações com o tempo, considera-se que mudanças podem ocorrer em três tipos de dados: nos conteúdos, na rede e nas requisições. Para simplificar o problema, o tempo foi discretizado em partes chamadas de períodos de tempo.

As mudanças na rede ocorrem através da mudança no atraso entre servidores. Claro que a mudança em um enlace da rede pode provocar mudança no custo de comunicação entre vários servidores, sendo necessário recalcular os custos para todos os servidores que usavam o enlace alterado para se comunicarem com outros servidores. Existem formas eficientes para recalcular estes custos, sem que haja necessidade de recalcular todos os custos de comunicação [17, 47]. Como estes cálculos não fazem parte do escopo deste

trabalho considerou-se que o custo de comunicação entre cada par de servidores afetados será informado quando ocorrer uma mudança na rede.

Uma outra característica do problema que também pode ser tratada de maneira dinâmica é o surgimento e remoção de conteúdos dentro da RDC. Quando o fornecedor de conteúdo faz uma atualização em um conteúdo pequeno, normalmente as réplicas desatualizadas espalhadas pela RDC são descartadas e o conteúdo atualizado é baixado pelos servidores da RDC a partir do fornecedor. Esta política pode ser adotada porque os arquivos solicitados são geralmente pequenos, fazendo com que sua transferência seja praticamente instantânea em ambientes de rede de grande velocidade.

Para arquivos multimídia, como vídeos e áudios por exemplo, as alterações são normalmente a substituição de um arquivo por outro. Como estes arquivos geralmente são muito extensos, sua transferência não é imediata, dispendendo às vezes muito tempo para ser concluída. Esta demora, juntamente com o uso da estratégia de descarte de réplicas para estes arquivos, pode causar uma sobrecarga no servidor, visto que as requisições serão reencaminhadas para ele à medida que as réplicas forem removidas, causando assim, uma queda na qualidade.

A partir da observação de softwares de distribuição de mídias amplamente difundidos, contatou-se que quando uma alteração é feita em um conteúdo e o conteúdo alterado é disponibilizado na rede, as réplicas antigas não são descartadas. O que de fato ocorre é que o conteúdo alterado é disponibilizado na rede como um novo conteúdo, que fica disponível juntamente com o antigo, cabendo ao cliente escolher qual deles quer receber. Esta abordagem para conteúdos extensos é bastante vantajosa, visto que não prejudica os atendimentos em andamento, uma vez que as réplicas do conteúdo solicitado não são descartadas, além de possibilitar que as novas requisições recebam o conteúdo atualizado. À medida que os atendimentos às requisições que solicitaram o conteúdo antigo forem sendo terminados, o número de réplicas do conteúdo antigo pode ir diminuindo até que não seja mais necessário manter uma réplica do conteúdo antigo.

Assim, para este trabalho, optou-se por modelar o surgimento e remoção de conteúdos da RDC através da atribuição de um tempo de vida para cada conteúdo. Quando um conteúdo for substituído por um novo, ele receberá uma marcação dizendo que não existirão novas requisições para aquele conteúdo, fazendo assim com que as réplicas deste conteúdo sejam descartadas à medida que os atendimentos em andamento forem terminando. O período em que o atendimento a última requisição é concluído é chamado de período final da vida de um conteúdo e o período em que este é inserido na RDC é chamado de período

inicial.

Para modelar a dinâmica de requisições considerou-se que estas podem surgir e terminar dentro de um horizonte de planejamento. Uma requisição possui demanda por um conteúdo, mas esta demanda não é persistente. Uma vez que o conteúdo tenha sido obtido pelo cliente, a requisição é encerrada, o que caracteriza o fim do atendimento. O mais lógico a se pensar, nesse caso, é que a requisição possui demanda por um conteúdo por um determinado número de unidades de tempo. Em ambientes de rede com QoS, uma qualidade mínima é exigida pelas requisições, contudo, geralmente não há imposições sobre a qualidade máxima. Isto leva a um ponto chave da discussão sobre as demandas. Se o sistema da RDC for mantido de modo que todas as requisições tenham somente seus requisitos mínimos atendidos, ele (o sistema de RDC) pode estar sendo subutilizado.

Garantir os requisitos mínimos não quer dizer que o sistema da RDC não pode oferecer mais qualidade aos seus usuários. Alguns servidores podem ter banda sobrando e capacidade de atender ainda melhor os clientes. Por exemplo, uma requisição que exige uma banda mínima de 5 KB por unidade de tempo para um conteúdo de 100 KB é atendida por um servidor que tem, no momento, 50 KB por unidade de tempo de banda não utilizada. Se o sistema fosse mantido de modo que somente as exigências mínimas fossem atendidas, o servidor atenderia a requisição com apenas 5 KB de sua banda, levando exatamente 20 unidades de tempo para finalizar o atendimento. Porém, se o sistema pudesse enviar mais que o mínimo, apenas duas unidades de tempo seriam necessárias. Contudo, para prover este tipo de atendimento otimizado, é necessário que o servidor conheça o limite máximo da banda da requisição. Por exemplo, não faz sentido o servidor enviar o conteúdo utilizando os seus 50 KB de banda excedente se o máximo que a conexão do cliente que enviou a requisição suporta é de 10 KB. Considerar este tipo de atendimento otimizado torna o problema mais realista e abre um grande leque de possibilidades para trabalhos futuros.

Tendo esta questão do atendimento otimizado em vista, a divisibilidade da demanda se torna ainda mais importante. Com o uso de mais de um servidor para atender uma requisição, as chances de atendê-la em um menor tempo aumentam, além de proporcionar o já mencionado benefício de tolerância a falhas.

Ao se pensar em um caso de atualização/escrita para o PPRDR, tem-se a impressão de que este caso não apresenta maiores dificuldades, assim como mostrado em [8]. Contudo, a inclusão de um caso de escrita (mudança ou atualização no conteúdo) não pode ser tratada de maneira trivial. Para requisições que ainda não chegaram, este tratamento se

torna trivial, bem como o tratamento para as requisições que já foram atendidas, basta simplesmente passar a estas requisições o conteúdo atualizado. O caso crítico se encontra nos atendimentos que estão em andamento.

Na abordagem tradicional para as RDC, réplicas desatualizadas são simplesmente descartadas [8, 50]. Esta estratégia é potencialmente desvantajosa para o PPRDR abordado neste trabalho, pois descartar uma réplica significa realocar todas as requisições que estavam sendo atendidas por ela para uma outra réplica, fato este que pode sobrecarregar um dos servidores da RDC e/ou causar uma queda na qualidade percebida pelos clientes. Nos trabalhos citados, os autores consideram que um período de tempo é suficientemente extenso para atender todas as requisições pendentes. Isto é possível devido ao fato de que os conteúdos tratados nestes trabalhos são geralmente pequenos (até dois Megabytes), e os períodos de tempo suficientemente grandes para que todas as requisições pendentes sejam atendidas. Assim, pode-se considerar que as requisições não se propagam de um período para outro, o que acontece é o surgimento de novas requisições.

Tais suposições, porém não podem ser feitas para arquivos grandes (vídeos, por exemplo) que podem ter tamanhos variando entre alguns Kilobytes e alguns Gigabytes. Frequentemente, estes conteúdos consomem muito tempo para sua transferência, o que torna irreal a possibilidade de supor um intervalo de tempo suficientemente grande para que todas as requisições sejam atendidas. Também há o fato que não há como prever quando as requisições irão chegar. Isso implica que se uma requisição chegar no meio de um intervalo de tempo ela terá que esperar até o próximo intervalo para começar a ser atendida. Caso estipule-se um intervalo de tempo muito grande, uma requisição que se encontra na situação de espera pode ter que aguardar durante muito tempo. Por isso, o ideal é que os períodos de tempo sejam tão curtos quanto possível, fazendo com que as requisições se propaguem no tempo e diminuindo o tempo de espera para que uma requisição comece a ser atendida.

Para os atendimentos em andamento, há ainda outro caso problemático, que é o de alteração no meio de um conteúdo. Suponha que um vídeo de 300 quadros seja disponibilizado na RDC e que por algum motivo, depois de algum tempo já disponível ele sofra uma edição no quadro 200. Para os usuários que ainda não baixaram este quadro não há problemas, mas, o que fazer com os clientes que já baixaram este quadro? Em alguns tipos de mídia, os quadros são descritos em função dos quadros anteriores, logo, simplesmente reenviar o quadro alterado pode acarretar problemas no momento da visualização, provocando perda de qualidade.

Observando programas amplamente difundidos na atualidade, percebe-se que, ao editar um conteúdo e disponibilizá-lo na rede, o conteúdo antigo permanece, e o conteúdo editado surge como novo conteúdo. Assim, cabe ao cliente escolher qual conteúdo ele quer receber.

Incorporar esta abordagem de criar novos conteúdos em uma RDC é extremamente simples, visto que as requisições pedem por um conteúdo sem saber seu nome real, cabendo ao servidor mapear o nome lógico, fornecido pelo cliente, para o nome real, presente nos servidores. Como visto anteriormente no Capítulo 2 este mapeamento é feito através de uma cadeia de DNS. Contudo, não é trivial modelar isso matematicamente. Uma idéia é dar a uma réplica um tempo de vida. No caso real, o que ocorre é que um conteúdo desatualizado deixa de receber requisições e, possivelmente, recebe uma marcação como estando desatualizado, podendo ser removido quando houver necessidade. Contudo, os atendimentos em andamento não são interrompidos, fazendo assim com que não haja comprometimento da qualidade do conteúdo recebido.

Para se calcular o tempo de vida de um conteúdo, basta então calcular o período que o último cliente, dentre os que estão sendo atendidos atualmente, será atendido. Tendo informações sobre a banda mínima do cliente, o tamanho do conteúdo, e a fração entregue até o presente momento, este cálculo se torna trivial.

Para ambientes com alto grau de imprevisibilidade, o uso de modelos dinâmicos *offline* não é totalmente adequado, visto que, estes modelos consideram a existência de dados sobre a demanda futura e sobre todas as mudanças que ocorrerão, tirando muitas vantagens destes dados. Supor a existência de tais dados em um ambiente de redes não é razoável, uma vez que não há como prever o volume de demandas nem sua origem, bem como não é possível fazer previsões sobre as mudanças nos enlaces e no surgimento de conteúdos. Entretanto, os modelos *offline* são capazes de resolver bem os problemas e fornecem um bom parâmetro de comparação para os modelos *online*.

Os modelos *online*, onde os dados chegam ao longo do tempo, parecem mais adequados aos ambientes com alto grau de imprevisibilidade como o ambiente apresentado por uma RDC. Contudo, nada impede que os modelos *online* façam uso de algoritmos adaptativos, que tentam prever a demanda futura baseada na demanda passada.

Assim sendo, as características do PPRDR que é abordado neste trabalho são:

- Servidores Capacitados - existência de limites, heterogêneos, de banda e espaço em disco nos servidores.

- Multiproduto - existência de mais de um conteúdo na rede.
- Dinâmico - mudanças na rede, nos conteúdos e nas demandas ocorrem ao longo tempo.
- *Online* - os dados chegam ao longo do tempo e nada se pode supor sobre os dados futuros.
- Múltiplo atendimento - possibilidade de atendimento de uma demanda por mais de um servidor.
- Demanda fracionária - possibilita a divisão de conteúdos em Kilobytes ou Megabytes.
- QoS - presença de requisitos mínimos de banda e atraso nas requisições
- Atendimento otimizado - tentativa de atender melhor os clientes sempre que possível, utilizando limites máximos nas requisições.

Existem diversos trabalhos relacionados à RDCs que tratam de problemas similares ao PPRDR abordado nesta tese. O próximo capítulo apresenta resumidamente alguns destes trabalhos.

Capítulo 4

Trabalhos Relacionados

Embora do conhecimento dos autores não exista nenhum trabalho na literatura sobre o modelo do PPRDR como enfocado nesta tese, existem vários trabalhos na literatura que tratam de temas relacionados à RDC. Diversas metodologias foram usadas para tratar os problemas gerenciais e técnicos, dentre as quais destacam-se as abordagens exatas através do uso de formulações matemáticas e as abordagens heurísticas através de algoritmos construtivos e meta-heurísticas.

Em [14], os autores tratam vários problemas relacionados à RDC, incluindo o Problema de Dimensionamento e Posicionamento de Servidores e o PPRDR. 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 Bender que consiste em fixar os valores de algumas das variáveis, criando assim, um subproblema menor que o anterior mas mantendo a viabilidade da solução. Um algoritmo heurístico guloso também é proposto pelos autores. 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 decrescente de capacidade de armazenamento dos mesmos. Após ativar um servidor, os problemas de distribuição de requisições e de replicação são resolvidos levando em conta todos os servidores ativos da RDC. O primeiro é resolvido através da associação de um cliente com o servidor que resulta no menor custo de comunicação. O segundo é resolvido através do cálculo das "economias de conteúdos" que representam o benefício de replicar um conteúdo em um servidor. Após calculado este benefício para cada conteúdo, os conteúdos são posicionados no servidor de acordo com uma ordem não crescente destes benefícios sem violar as restrições de capacidade de armazenamento dos servidores. Os resultados computacionais mostraram que os algoritmos propostos atingiram bons resultados, alcançando uma

diferença de no máximo 5% em relação à solução ótima.

Huang *et al.* [25] afirmam que apesar do número significativo de trabalhos relacionados com RDC, pouca atenção foi dada às questões relacionadas com a QoS percebida pelos clientes. Os autores tentam dar garantias de baixos atrasos usando uma noção relaxada de garantia que seria vista como uma alta chance de sucesso. Na abordagem proposta, o fornecedor do conteúdo negocia com o provedor de RDC os parâmetros de QoS que devem ser atingidos para um conteúdo, criando assim, um mecanismo para que conteúdos diferentes possam ter parâmetros de QoS diferentes. Os parâmetros de QoS são ajustados de acordo com o tamanho dos conteúdos, permitindo ao sistema classificar um conteúdo em categorias amplamente abrangentes de acordo com os parâmetros de QoS estabelecidos. Uma modelagem por grafos é utilizada e os algoritmos propostos precisam ser executados uma vez para cada categoria de conteúdos criada. Uma abordagem distribuída baseada na estratégia de dominação de grafos é proposta para o Problema de Posicionamento de Réplicas estático. O objetivo é construir um grafo dominante mínimo no qual os vértices serão os servidores escolhidos para posicionar as réplicas. O algoritmo proposto foi comparado com outros algoritmos de dominação de grafos, mostrando que a abordagem distribuída consegue encontrar soluções que apresentam um custo de replicação muito baixo e que também respeitam os parâmetros de QoS estabelecidos.

Em [9], os autores lidam com uma variação do Problema de Posicionamento de Réplicas estático. Os autores propõem o uso de uma árvore de *multicast* para entregar os conteúdos para os clientes finais ao invés do uso dos caminhos mínimos, tradicionalmente usados nos fluxos *unicast*, argumentando que o uso de fluxos *unicast* produz soluções que não são os ótimos globais para os problemas de fluxo de transmissão escaláveis. São propostos uma formulação matemática e seis heurísticas para o problema. As heurísticas propostas, são combinações de dois conjuntos de heurísticas para problemas menores, duas para posicionamento de réplicas e três para a construção das árvores de *multicast*. Estas heurísticas para problemas menores trabalham em conjunto de maneira intercalada, inserindo novas réplicas e calculando árvores de *multicast* a cada iteração baseadas na configuração atingida na iteração anterior. Os resultados apresentados mostram que as heurísticas produziram bons resultados em tempos computacionais consideravelmente inferiores aos tempos de resolução da formulação matemática proposta. Apesar de levar em conta o custo de entrega de um conteúdo pela RDC, a qualidade percebida pelo cliente final não é levada em consideração neste trabalho. Outra limitação das propostas do referido trabalho deve-se ao fato das abordagens propostas não consideram a entrega de múltiplos conteúdos.

Em [45] os autores 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.

Em [52], são abordados o Problema de Replicação de Conteúdos e o Problema de Posicionamento de Réplicas. Três algoritmos de replicação são propostos. O primeiro cria réplicas iterativamente baseado nos custos de comunicação. A cada iteração, o algoritmo cria uma réplica do conteúdo que tem o maior custo de comunicação. Em seguida, um novo custo de comunicação é calculado usando o número de réplicas atualizado e passado para a próxima iteração. O algoritmo continua criando réplicas seguindo esta política até que a capacidade de armazenamento do *cluster* seja exaurida. Segundo os autores, a desvantagem deste algoritmo é a sua complexidade devido ao fato de que ele deve calcular a capacidade de replicação do *cluster*. O segundo algoritmo proposto determina o número de réplicas baseado na popularidade do conteúdo. Através de um conhecimento *a priori* a respeito da popularidade dos conteúdos, este algoritmo calcula heuristicamente o número de réplicas para cada conteúdo partindo do pré-suposto de que a popularidade dos conteúdos segue uma distribuição do tipo Zipf [54]. O terceiro cria as réplicas seguindo uma estratégia *round-robin* com pesos baseada na popularidade dos conteúdos, sendo que os conteúdos mais populares são replicados mais vezes. Para o posicionamento das réplicas, dois algoritmos são propostos. Para o primeiro, os autores afirmam que se os algoritmos de replicação levarem ao fato de que todas as réplicas tenham um custo de comunicação uniforme, então um algoritmo *round-robin* resultará na solução ótima, contudo, sabe-se que os custos de comunicação não são uniformes e, por isso, os autores propuseram um segundo algoritmo que posiciona réplicas com grande custo de comunicação em servidores com menor carga, garantindo que servidores não tenham mais de uma réplica do mesmo conteúdo. Os autores também propõem uma heurística *Simulated Annealing* [26] para uma generalização do problema onde um mesmo conteúdo pode ter vários graus de qualidade. Os algoritmos foram testados em um cenário composto para distribuição de vídeos, com servidores tendo grande capacidade de armazenamento, grande capacidade de banda e com conteúdos extensos (90 minutos - 2 GB). Os resultados obtidos mostram que os algoritmos propostos conseguiram soluções de alta qualidade sendo que, em alguns casos,

as soluções ótimas foram encontradas. Uma das limitações do referido trabalho reside no fato de que os algoritmos nele propostos consideram dados conhecidos *a priori* sobre a popularidade dos conteúdos, dados estes difíceis de se obter na prática.

Em [13], os autores lidam com o PPR Dinâmico. Os autores modelam o problema como um processo de Markov, o qual é capaz de lidar com sistemas dinâmicos usando distribuições matemáticas. Requisições são modeladas como uma taxa de entrada, representando quantas requisições o sistema recebe por unidade de tempo o que permite modelar a dinâmica do sistema. Usando taxas de chegada para as requisições e um estado inicial de posicionamento de réplicas, encontrado no período anterior, o problema de decisão se torna encontrar a melhor mudança a ser feita no estado de posicionamento de réplicas para atender, da melhor forma possível, a demanda do período atual. Analisando o comportamento das soluções ótimas encontradas pelo processo de Markov, os autores elaboraram uma heurística para ser usada em instâncias maiores para o problema. A heurística é um algoritmo guloso que associa novas requisições no conjunto de réplicas atual. Caso um crescimento nas requisições possa ser acomodado pela conjuntura atual, o algoritmo procura uma réplica para remover. Caso contrário, o algoritmo procura um servidor para inserir uma nova réplica. Os resultados obtidos mostram que a heurística obteve resultados muito próximos aos ótimos. Apesar de não tratar explicitamente de requisitos de QoS, este trabalho considera que uma requisição não pode ser atendida por servidores que estão a uma distância maior que um certo limiar d, que também é uma constante investigada no trabalho. Como a distância em [13] é uma medida genérica, não permitir que servidores que se encontram a uma distância maior que d atendam uma determinada requisição pode ser visto como uma exigência na qualidade do serviço prestado e, assim sendo, este limiar de distância é tratado como restrição de QoS nesta tese.

Em [49], os autores utilizam conceitos de redes P2P para resolver o PPR. Assim como nas redes P2P, os servidores comunicam-se entre si, trocando informação sobre a carga de cada um e sobre o tráfego local, utilizando estas informações para, localmente, tomar decisões de criar, apagar ou migrar réplicas, tornando a infra-estrutura da RDC mais robusta, eficiente e tolerante a falhas. Duas heurísticas são propostas. Uma que leva em conta apenas os custos relativos ao transporte dos conteúdos e uma que leva em conta o custo do transporte e o custo de armazenamento. Os autores também fazem uma discussão a respeito do uso de mecanismos de balanceamento de carga em arquiteturas de RDC mostrando que o uso de tais mecanismos, além de promover o aumento médio do uso de cada enlace da rede, reduz a sobrecarga nos servidores. Apenas os resultados de uma das heurísticas foram mostrados. Esta heurística, que leva em conta apenas o

custo de transporte dos conteúdos, foi comparada com uma série de outras heurísticas de posicionamento de réplicas existente mostrando que a heurística proposta obteve a melhor relação custo-benefício em termos de qualidade de solução e tempo de execução.

Em Coppens *et al.* [18], uma arquitetura híbrida (centralizada/distribuída) é proposta para resolver o PPR Dinâmico com o objetivo de minimizar o tráfego na rede. Segundo os autores as abordagens centralizadas podem atingir o posicionamento ótimo, porém, estas estratégias não são escaláveis para redes de grandes dimensões. Na abordagem proposta, todos os servidores executam uma cópia do algoritmo de posicionamento levando em conta apenas dados locais e algumas informações enviadas periodicamente pela arquitetura da RDC. Depois disso, cada servidor procura replicar os conteúdos de maior custo de comunicação, trazendo para si os conteúdos mais desejados pelos clientes ligados a este servidor. Caso o servidor não tenha espaço suficiente para armazenar tal réplica, ele procurará, dentre as réplicas que ele contém, aquela que traz menor benefício para ele, e trata esta réplica como uma candidata à remoção. Se o benefício trazido pela réplica candidata à remoção for menor que o benefício trazido pela réplica a ser inserida, o conteúdo de menor benefício é removido em favor do que trará maior benefício. Esta abordagem apresenta resultados que são compatíveis com as abordagens centralizadas e ainda mantém as qualidades de uma abordagem distribuída em relação à escalabilidade e tolerância a falhas.

Em [46], são propostas: uma abordagem distribuída e uma centralizada para resolver o PPR Dinâmico. Os algoritmos levam em consideração taxas calculadas com base nos custos de comunicação, processamento e na frequência de acesso às réplicas. Na abordagem centralizada, um servidor central recolhe dados e executa um algoritmo guloso de posicionamento de réplicas que leva em consideração os dados coletados. Na abordagem distribuída, os servidores mantêm dados relativos à latência e frequência de acesso aos conteúdos do próprio site e dos sites vizinhos. Testes são feitos periodicamente para decidir sobre a remoção ou criação de réplicas. As decisões são tomadas localmente de acordo com as informações armazenadas em cada servidor. Os algoritmos propostos são comparados entre si tendo o distribuído apresentado melhores resultados.

Em [8] uma versão dinâmica do PPRDR é abordada, sendo contextualizado 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. As soluções obtidas são comparadas com um outro algoritmo proposto para o mesmo problema chamado *ACDN*, mostrando que as soluções encontradas pelos métodos propostos superam as do *ACDN* e que as soluções encontradas pela heurística estão próximas da solução ótima.

Em [50] os autores apresentam uma solução dinâmica e distribuída para o problema de replicação de um objeto em um sistema distribuído, que é um caso particular do problema abordado nesta proposta de tese. O algoritmo proposto tem por objetivo caminhar progressivamente para o ótimo global, mudando de um esquema de replicação para outro, de acordo com os padrões de leitura e escrita, contudo, este algoritmo consegue garantir a descoberta da solução ótima somente para redes com topologia em árvores, sendo incapaz de garantir um ótimo global para topologias descritas por grafos. Os autores mostram que apesar de ser um algoritmo distribuído, o método proposto requer muito menos mensagens para sua execução do que a quantidade de mensagens necessária para centralizar os dados, utilizar uma abordagem centralizada e retransmitir a solução encontrada. Periodicamente, os processos realizam testes baseados no número de operações de leitura e escrita para, localmente, decidir se vão manter, remover ou adquirir réplicas do objeto em questão. Os autores também fazem uma série de discussões a respeito de aspectos práticos do método tais como necessidade de manter um conjunto conexo de réplicas, a periodicidade de execução, discrepância entre o número de leituras e escritas, etc. Os autores também propõem um método formal de análise de algoritmos de replicação adaptativos mostrando que o algoritmo proposto atinge um ótimo para padrões de escrita e leitura regulares. O algoritmo proposto também foi testado por simulação mostrando que este realmente apresenta um custo de comunicação menor do que uma abordagem centralizada para o problema. Uma das limitações deste trabalho é o fato de não considerar nada além do número de leituras e escritas para mudar o esquema de replicação. Questões como a carga nos servidores, o custo de transitar um objeto pela rede no momento da replicação e a qualidade de atendimento das requisições não são consideradas pelos autores.

A Tabela 4 mostra alguns dos trabalhos considerados, pelos autores, mais sofisticados em termos de abordagens. Em comum todos estes trabalhos tem o fato de utilizar mais de um tipo de metodologia para resolver o problema trabalhado. Outro ponto em comum entre estes trabalhos é que todos eles tratam de múltiplos conteúdos simultaneamente. A primeira linha da tabela indica o(s) problema(s) tratados em cada um dos trabalhos. Note que diversos problemas podem ser trabalhados dentro do contexto de uma RDC. A

segunda linha indica a versão do problema trabalhado. A terceira linha apresenta, para cada um dos trabalhos, os pontos onde o problema se mostra dinâmico. Note que na maioria dos trabalhos apenas as requisições são dinâmicas. A quarta linha mostra um diferencial deste trabalho que é o fato de cada requisição poder ser diferente das outras. Todos os trabalhos listados tratam requisições por um mesmo conteúdo originarias de um mesmo servidor como sendo requisições indistinguíveis. Neste trabalho, como cada uma das requisições possui atributos de QoS diferentes, elas não podem ser tratadas de maneira homogênea. A quinta linha mostra a existência (ou ausência) de conteúdos fixos. Nos trabalhos onde existem conteúdos fixos, não existe a preocupação de remover todas as réplicas de um conteúdo, visto que todos os conteúdos possuem uma réplica permanente em algum local da RDC. A existência de cenários assimétricos nas análises dos trabalhos é apresentada na sexta linha. As linhas sete e oito mostram como são as capacidades do servidores. A nona linha mostra como a QoS é tratada nos diversos trabalhos. Note que entre os trabalhos da tabela apenas dois se preocupam com restrições de QoS. A décima linha mostra outro diferencial deste trabalho que é o atendimento simultâneo por múltiplos servidores. A décima primeira linha mostra quais os trabalhos que possuem preocupação com o balanceamento de carga nos servidores. A penúltima linha mostra os tipos de custo analisados em cada um dos trabalhos. Estes custos estão representados com expressões de letras que possuem o seguinte significado: A letra "A" representa a disponibilidade do sistema; A letra "C" representa o custo por uso de servidores; O custo por atraso nas requisições é representado pela letra "D"; A letra "H" representa o custo de antedimento das requisições; A letra "L" representa o custo por desbalanceamento do sistema e o custo de replicação dos conteúdos é representado pela letra "R". As metodologias de cada trabalho estão expostas na última linha da tabela onde a letra "E" indica o uso de uma metodologia exata, a letra "H" indica o uso de uma metodologia heurística, a letra "Y" indica o uso de metodologias Híbridas e a letra "M" indica o uso de meta-heurísticas.

Características	[14]	[13]	[52]	[8]	Este Trabalho
Problema Tratado	PPS+PPRDR	PPR	PR+PPR	PPRDR	PPRDR
Versão do Problema	Estático	Din. Offline	Din. Offline	Din. Online	Din. Online
Dinamismo	-	Req.	Req.	Req.	Req.+Cont.+Rede
Requisições Heterogêneas	Não	Não	Não	Não	sim
Conteúdos fixos	Sim	Não	Não	Sim	Não
Cenários Assimétricos	Não	Não	Não	Não	Sim
Disco nos Servidores	Heterogêneo	Homogêneo	Homogêneo	Homogêneo	Heterogêneo
Banda nos Servidores	Infinita	Infinita	Homogênea	Homogênea	Heterogênea
QoS	-	Dist. Max.	-	-	Atraso /Banda.
Atend. Múltiplos Servidores	Não	Não	Não	Não	sim
Balanceamento de Carga	Não	Não	Sim	Sim	Não
Custos Analisados	C+H	H+U	A+L	H+R+U	H+D+R
Metodologia	E+H	E+H	H+M	E+H	E+Y+H+M

Tabela 4.1: Trabalhos Relacionados

Capítulo 5

Abordagens Exatas

Tendo observado o uso de métodos exatos nos trabalhos relacionados e já tendo feito alguns estudos preliminares sobre o tema [37, 39] , são propostas, nesta tese, algumas formulações matemáticas considerando várias características do problema e também um algoritmo enumerativo. Apesar do fato de que as formulações matemáticas não serem adequadas para a resolução de modelos *online* seu uso ainda sim é importante para obtenção de valores ótimos e/ou limites, os quais podem ser usados como guias para as métodos heurísticos. Assim, as formulações apresentadas neste capítulo são utilizadas como métodos para resolução das versões *offline* do PPRDR e os resultados obtidos pelas mesmas são usados como guias para as demais abordagens. Vale a pena lembrar que na versão *offline*, todas as mudanças são conhecidas a priori, fato este que possibilita uma alocação muito mais eficiente das réplicas, reduzindo assim, os custos operacionais.

Assim, algumas definições devem ser feitas agora e consideradas para o restante deste capítulo.

Seja R o conjunto de requisições a serem atendidas, S o conjunto de servidores da RDC, C o conjunto de conteúdos a serem replicados. Sejam também as constantes $origem(i)$ o servidor de origem da requisição i, $ld(i)$ o atraso local da requisição i, $atraso(j_1, j_2)$ o atraso entre dois servidores e $RTT(j_1, j_2)$ o tempo que uma mensagem leva para percorrer o caminho de um servidor j_1 para um outro servidor j_2 e voltar ao primeiro (*Round Trip Time* - RTT). Além disso, Sejam L_k o tamanho do conteúdo k, $G(i)$ o conteúdo exigido pela requisição i, BR_i a banda exigida pela requisição i, $atrasoMax(i)$ o atraso máximo permitido pela requisição i, AS_j o espaço em disco disponível no servidor j e MB_j a banda máxima do servidor j. Estes conjuntos e constantes são usados durante todo o capítulo e por isso devem estar sempre claros.

As formulações apresentadas neste capítulo são o resultado de um processo incremental de inclusão de requisitos ao problema. Deste modo, as primeiras formulações tratam de versões bem simplificadas e foram sendo aprimoradas ao longo dos estudos que resultaram nesta tese. Assim sendo, a primeira formulação trata de uma versão bem simples do problema uma vez que ela não permite divisibilidade das demandas e também pelo fato de tratar do problema estático ao invés do problema dinâmico, que é o alvo deste trabalho. Entretanto, por questões didáticas, optou-se por incluir esta formulação para que o processo de aprimoramento das formulações desenvolvidas pudesse ser melhor compreendido.

5.1 Formulação para o PPRDR Estático

Para esta primeira formulação mais algumas definições devem ser feitas. NR_k o número máximo de réplicas permitido para um conteúdo k e c_{ij} o custo de atendimento da requisição i pelo servidor j.

Definimos as seguintes variáveis e notações

Variáveis:

- $y_{jk} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está replicado no servidor } j \\ 0 & \text{caso contrário} \end{cases}$
- $x_{ij} = \begin{cases} 1 & \text{se o requisição } i \text{ é atendido pelo servidor } j \\ 0 & \text{caso contrário} \end{cases}$

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- L_k o tamanho do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- BR_i exigência de banda da requisição i.

- NR_k número de réplicas disponíveis para o conteúdo k.
- $G(i)$ o conteúdo exigido pela requisição i.
- c_{ij} custo de atendimento da requisição i no servidor j, calculado pela seguinte equação $c_{ij} = (RTT_{j,origem(i)} + atraso_{j,origem(i)} + ld(i)) \times BR_i$.

Toda vez que o atraso entre um servidor j e o servidor origem da requisição juntamente com o atraso local da requisição for maior que o atraso máximo permitido pela requisição, é somado ao custo de atendimento de j uma penalidade dada por $\rho + \rho \times (atraso_{j,origem(i)} + ld(i) - atrasoMax(i))$. Onde ρ é uma constante maior ou igual a 1. Neste trabalho usou-se $\rho = 1000$ para todas as formulações e heurísticas.

É importante fazer algumas considerações a respeito desta formulação para melhor compreensão. Para fins de simplificação, considerou-se que, inicialmente, todos os conteúdos estariam posicionados no mesmo servidor, possivelmente o servidor principal da RDC, e então replicados para os demais servidores. Também foi proposta a existência de um limite no número máximo de réplicas para um conteúdo, possivelmente estabelecido contratualmente, aqui referenciada através da constante NR_k. Assim, a seguinte formulação é proposta para a versão estática do PPRDR.

$$Min \quad \sum_{i \in R} \sum_{j \in S} c_{ij}\, x_{ij} + \sum_{j \in S} \sum_{k \in C} L_k\, y_{jk} \tag{5.1}$$

$S.a.$

$$\sum_{j \in S} x_{ij} = 1,\ \forall i \in R, \tag{5.2}$$

$$\sum_{i \in R} BR_i\, x_{ij} \ \leq \ MB_j, \forall j \in S, \tag{5.3}$$

$$x_{ij} \leq y_{j\,G(i)}, \forall i \in R, \forall j \in S, \tag{5.4}$$

$$\sum_{j \in S} y_{jk} \ \leq \ NR_k, \forall k \in C, \tag{5.5}$$

$$\sum_{k \in C} L_k\, y_{jk} \ \leq \ AS_j, \forall j \in S, \tag{5.6}$$

$$x_{ij} \in \{0,1\}, \forall i \in R, \forall j \in S, \tag{5.7}$$

$$y_{jk} \in \{0,1\}, \forall j \in S, \forall k \in C. \tag{5.8}$$

Na formulação *FE1*, descrita entre (5.1) e (5.8), a função objetivo mostrada em (5.1) requer a minimização do consumo total de banda e também a redução dos custos de

migração. As restrições (5.2) exigem que cada uma das as requisições seja atendida por um único servidor. As restrições (5.3) controlam a capacidade da banda de saída dos servidores. As restrições (5.4) impedem que uma requisição seja atendida por um servidor que não possui uma réplica do conteúdo exigido pela requisição. As restrições (5.5) limitam o número máximo de réplicas por conteúdo. As restrições (5.6) controlam o espaço em disco dos servidores e as restrições (5.7) e (5.8) são as restrições de integralidade e não negatividade.

Esta formulação para a versão estática do PPRDR é resultado de estudos preliminares sobre o tema [37, 39] e pode ser usada em casos onde a demanda seja previsível. Assim, dada uma certa previsão de demandas, a formulação estática pode ser usada para um posicionamento inicial das réplicas de conteúdos vindos de um mesmo provedor.

5.2 Formulação para o PPRDR Dinâmico com Demanda Divisível

A partir desta formulação e de análise dos trabalhos relacionados, é proposta uma primeira formulação para a versão dinâmica do PPRDR, que é baseada na formulação para o Problema de Localização de Facilidades com abertura, fechamento e reabertura de facilidades, proposta por Dias *et al.* e apresentada em [19].

A formulação, proposta neste trabalho, difere da mostrada em [19] em dois aspectos: Primeiro, a formulação proposta por Dias *et al.* trata de maneira diferente a abertura e a reabertura de facilidades. Isto ocorre devido à uma diferenciação nos custos relativos à infra-estrutura necessária para se abrir uma facilidade como, por exemplo, construção de galpões, instalação de rede elétrica, compra de terreno, etc. Estes custos não aparecem no processo de reabertura de facilidades, visto que, muito da infra-estrutura deixada no momento do fechamento permanece no local. Para o PPRDR, considerando cada servidor como uma facilidade e que abrir uma facilidade significa replicar um conteúdo, não há diferença de entre os custos de abertura e reabertura de uma facilidade visto que o custo a ser pago nos dois casos é o mesmo.

A segunda diferença é que a formulação proposta neste trabalho é adaptada para múltiplos produtos, característica não tratada em [19]. No contexto do PPRDR, cada produto é associado a um conteúdo. Como o problema trata de múltiplos conteúdos o modelo necessariamente deve abordar múltiplos produtos.

Assim para esta nova formulação, as seguintes notações foram definidas.

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t.
- $y_{kjuv} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está replicado no servidor } j \text{ nos períodos de } u \text{ até } v. \\ 0 & \text{caso contrário} \end{cases}$

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- T conjunto de períodos de tempo.
- L_k o tamanho do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- BR_{it} limite inferior de banda (exigência) do cliente i no período t.
- BX_{it} limite superior de banda (capacidade) do cliente i no período t.
- $G(i)$ o conteúdo exigido pela requisição i.
- c_{ijt} custo de atendimento da requisição i no servidor j, calculado pela seguinte equação $c_{ijt} = (RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times BR_{it}$.

As constantes BR_{it} e BX_{it} além de modelar a QoS exigida e a capacidade máxima para um cliente, também modelam a variabilidade da demanda. Caso estas duas constantes possuam o valor zero, interpreta-se que a demanda ainda não chegou ou que ela já foi atendida.

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} \sum_{k\in C}\sum_{j\in S}\sum_{u\in T}\sum_{v=u}^{T} L_k y_{kjuv} \tag{5.9}$$

S.a.

$$\sum_{j\in S} L_{G(i)}x_{ijt} \leq BX_{it},\ \forall i \in R, \forall t \in T, \tag{5.10}$$

$$\sum_{j\in S} L_{G(i)}x_{ijt} \geq BR_{it},\ \forall i \in, R \forall t \in T, \tag{5.11}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq MB_j, \forall j \in S, \forall t \in T, \tag{5.12}$$

$$\sum_{j\in S}\sum_{t\in T} x_{ijt} = 1,\ \forall i \in R, \tag{5.13}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{G(i)juv} \geq x_{ijt},\ \ \forall i \in R,\ \forall j \in S, \forall t \in T, \tag{5.14}$$

$$\sum_{k\in C}\sum_{u=1}^{t}\sum_{v=t}^{|T|} L_k\ y_{kjuv}\ \leq\ AS_j, \forall j \in S, \forall t \in T, \tag{5.15}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{kjuv} \leq 1,\ \forall j \in S,\ \forall k \in C,\ \forall t \in T, \tag{5.16}$$

$$x_{ijt} \in [0,1], \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.17}$$

$$y_{kjuv} \in \{0,1\}, \forall j \in S, \forall k \in C, \forall u, v \in T. \tag{5.18}$$

A função objetivo, exposta em (5.9), minimiza o custo de entrega dos conteúdos para os clientes. As restrições (5.10) impedem que o fluxo enviado a um cliente seja maior que a capacidade suportada por este. As restrições (5.11) são as restrições de QoS que indicam que o volume de fluxo entregue a uma requisição não pode ser inferior à banda exigida. As restrições (5.12) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (5.13) exigem que uma requisição seja plenamente atendida. As restrições (5.14) condicionam quem uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (5.15) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As restrições (5.16) impedem que uma réplica esteja alocada em um servidor em intervalos de tempo que se sobreponham. As demais restrições são as de integralidade e não negatividade.

Esta nova formulação contempla plenamente todas as restrições da versão dinâmica do PPRDR. Além disso, o fato de a variável x representar uma fração (e não uma deci-

são binária como na formulação anterior) permite a divisibilidade da demanda e facilita a resolução do problema uma vez que reduz o número de variáveis inteiras da formulação. Contudo, as questões de atualização de conteúdos e de cada conteúdo poder ter uma origem diferente não são contempladas. A questão do atendimento otimizado não é contemplada visto que a função objetivo só contempla os custos operacionais.

Nesta formulação não há limites contratuais para o número de réplicas de um conteúdo. Esta restrição foi removida após estudos de casos, feitos ao longo da tese, que constataram que os clientes de uma RDC não pagam pelo número réplicas e sim pela prestação do serviço. Assim, do ponto de vista do cliente, quanto mais réplicas existirem do seu conteúdo melhor.

5.3 Formulação para Atendimento Otimizado

Uma abordagem para modelar o atendimento otimizado é o uso da técnica de *backlog* [10]. Esta técnica permite que uma parte das demandas não seja atendida no devido momento, obrigando porém, que esta parte seja entregue no período seguinte.

Em termos gerais, uma demanda D_{it} por um produto i em um período t é igual a uma quantidade atendida no dia atual (x_{it}), menos o *backlog* do dia anterior ($b_{i(t-1)}$), mais o *backlog* do dia atual (b_{it}), que será pago no dia posterior. Assim temos que:

$$x_{it} - b_{i(t-1)} + b_{it} = D_{it} \tag{5.19}$$

Para uma formulação usando esta técnica, as seguintes variáveis e notações são definidas:

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t.
- $y_{kjuv} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está copiado no servidor } j \text{ nos períodos de} \\ & u \text{ até } v. \\ 0 & \text{caso contrário} \end{cases}$
- b_{it} *backlog* da requisição i no período t.

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- T conjunto de períodos de tempo.
- L_k o tamanho do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- D_{it} demanda da requisição i no período t.
- $G(i)$ o conteúdo exigido pela requisição i.
- c_{ijt} custo de atendimento da requisição i no servidor j, no período t, calculado pela seguinte equação $c_{ijt} = (RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times D_{it}$.
- p_{it} penalidade por usar *backlog* da requisição i no período t.

Assume-se que os períodos de tempo variam de 1 a $|T|$ e que o *backlog* para qualquer requisição no período $t = 0$ é nulo. É importante mencionar como é calculada a demanda D_{it} para uma requisição i. Primeiro, divide-se o tamanho do conteúdo solicitado por i pela banda máxima do cliente BX_i, obtendo assim, o número de períodos necessários para atender a requisição que é dado pelo quociente da divisão anterior. Assim, para cada período, após a chegada da requisição, a demanda D_{it} recebe a banda máxima da requisição i. Para o último período de atendimento, a demanda D_{it} recebe o resto da divisão de $L_{G(i)}/BX_i$.

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} + \sum_{i\in R}\sum_{t\in T} p_{it}b_{it} + \sum_{k\in C}\sum_{j\in S}\sum_{u\in T}\sum_{v=u}^{T} L_k y_{kjuv} \tag{5.20}$$

S.a.

$$\sum_{j\in S} L_{G(i)}x_{ijt} - b_{i(t-1)} + b_{it} = D_{it},\ \forall i \in R, \forall t \in T, \tag{5.21}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq MB_j, \forall j \in S, \forall t \in T, \tag{5.22}$$

$$\sum_{j\in S}\sum_{t\in T} x_{ijt} = 1,\ \forall i \in R, \tag{5.23}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{G(i)juv} \geq x_{ijt},\ \forall i \in R,\ \forall j \in S, \forall t \in T, \tag{5.24}$$

$$\sum_{k\in C}\sum_{u=1}^{t}\sum_{v=t}^{|T|} L_k\ y_{kjuv}\ \leq\ AS_j, \forall j \in S, \forall t \in T, \tag{5.25}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{kjuv} \leq 1,\ \forall j \in S,\ \forall k \in C, \forall t \in T, \tag{5.26}$$

$$x_{ijt} \in [0,1], \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.27}$$

$$y_{kjuv} \in \{0,1\}, \forall j \in S, \forall k \in C, \forall u, v \in T, \tag{5.28}$$

$$b_{it} \geq 0, \forall i \in R, \forall t \in T. \tag{5.29}$$

A função objetivo, exposta em (5.20), minimiza o custo de entrega dos conteúdos para os clientes bem como a quantidade de *backlog* feitos ao longo do tempo e os custos referentes à replicação dos conteúdos. As restrições (5.21) associam as variáveis x_{ijt} e b_{it}, dizendo que a soma das quantidades entregues no dia atual, mais a quantidade que será entregue no dia posterior é igual à demanda do dia atual mais o que se ficou devendo do dia anterior. As restrições (5.22) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (5.23) exigem que uma requisição seja plenamente atendida. As restrições (5.24) condicionam que uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (5.25) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As restrições (5.26) impedem que uma réplica esteja alocada em um servidor em intervalos de tempo que se sobreponham. As demais restrições são as de integralidade e não negatividade.

Esta formulação, chamada de *FB*, consegue fazer com que a quantidade máxima de

banda seja utilizada através de uma penalização. Contudo, o uso da técnica de *backlog* torna a formulação pouco natural e, às vezes, de difícil entendimento. Portanto, uma formulação matemática de mais fácil entendimento é proposta a seguir.

5.4 Formulação Com Penalização de Períodos

Buscando uma alternativa para o uso da técnica de *backlog*, que seja de mais fácil entendimento é proposta nesta seção uma nova formulação que segue as seguintes notações:

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t.
- $y_{kjuv} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está replicado no servidor } j \text{ no período de } u \text{ até } v. \\ 0 & \text{caso contrário} \end{cases}$
- dem_{it} representa a fração ainda não atendida da requisição i no período t.
- $db_{it} = \begin{cases} 1 & \text{se ainda há demanda de } i \text{ para ser atendida no período } t. \\ 0 & \text{caso contrário} \end{cases}$

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- T conjunto de períodos de tempo.
- L_k o tamanho do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- BR_{it} limite inferior de banda (exigência) do cliente i no período t.
- BX_{it} limite superior de banda (capacidade) do cliente i no período t.
- $G(i)$ o conteúdo exigido pela requisição i.

- A_i período de chegada da requisição i
- c_{ijt} custo de atendimento da requisição i no servidor j, no período t.
- cp_{it} custo fixo por atender uma requisição i no período t.

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} + \sum_{k\in C}\sum_{j\in S}\sum_{u\in T}\sum_{v=u}^{T} L_k y_{kjuv} + \sum_{i\in R}\sum_{t\in T} cp_{it}db_{it} \tag{5.30}$$

S.a.

$$\sum_{j\in S} L_{G(i)}x_{ijt} \leq BX_{it}\ db_{it},\ \forall i \in R, \forall t \in T, \tag{5.31}$$

$$\sum_{j\in S} L_{G(i)}x_{ijt} \geq BR_{it}\ db_{it},\ \forall i \in R, \forall t \in T, \tag{5.32}$$

$$dem_{it} = 1 - \sum_{j\in S}\sum_{z=A_i}^{t-1} x_{ijt}, \forall i \in R, \forall t\ inT, \tag{5.33}$$

$$db_{it} \geq dem_{it}, \forall i \in R, \forall t \in T, \tag{5.34}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq MB_j, \forall j \in S, \forall t \in T, \tag{5.35}$$

$$\sum_{j\in S}\sum_{t\in T|t\geq A_i} x_{ijt} = 1,\ \forall i \in R, \tag{5.36}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{G(i)juv} \geq x_{ijt},\ \forall i \in R,\ \forall j \in S, \forall t \in T, \tag{5.37}$$

$$\sum_{k\in C}\sum_{u=1}^{t}\sum_{v=t}^{|T|} L_k\ y_{kjuv}\ \leq\ AS_j, \forall j \in S, \forall t \in T, \tag{5.38}$$

$$\sum_{u=1}^{t}\sum_{v=t}^{|T|} y_{kjuv} \leq 1,\ \forall j \in S,\ \forall k \in C,\ \forall t \in T, \tag{5.39}$$

$$x_{ijt} \in [0,1], \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.40}$$

$$y_{kjuv} \in \{0,1\}, \forall j \in S, \forall k \in C, \forall u, v \in T, \tag{5.41}$$

$$dem_{it} \geq 0, \forall i \in R, \forall t \in T, \tag{5.42}$$

$$db_{it} \in \{0,1\}, \forall i \in R, \forall t \in T. \tag{5.43}$$

A função objetivo, exposta em (5.30), minimiza o custo de entrega dos conteúdos para os clientes bem como o custo de replicação e o número de períodos consecutivos a serem utilizados no atendimento. As restrições (5.31) e (5.32) dizem que se há demanda vinda da requisição i no período t, esta demanda deve ser suprida dentro de um padrão de qualidade

estabelecido. As restrições (5.33) estabelecem os valores das variáveis fracionárias dem_{it} para que nelas esteja contido a quantidade que ainda falta atender para cada demanda. As restrições (5.34) associam as variáveis dem_{it}e e db_{it} dizendo que, se a requisição i ainda não foi plenamente atendida no período t, então ela deve ser atendida neste período. As restrições (5.35) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (5.36) exigem que uma requisição seja plenamente atendida. As restrições (5.37) condicionam que uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (5.38) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As restrições (5.39) impedem que uma réplica esteja alocada em um servidor em intervalos de tempo que se sobreponham. As demais restrições são as de integralidade e não negatividade.

Esta formulação é mais natural para o problema, contudo, mesmo sabendo que uma formulação matemática serve apenas como limite para um problema *online*, pode-se perceber que esta formulação oferece um limite mais fraco do que a formulação *FB*. Isto ocorre porque esta formulação penaliza o número de períodos gastos no atendimento apenas através de um custo fixo. Um algoritmo guloso para o problema *online* tentaria atender o máximo possível de demanda a cada período, assim como a formulação *FB* faz. A formulação Proposta nesta seção tenta atender o máximo possível de demanda nos períodos de menor custo, sendo portanto apresentada apenas como ilustração.

5.5 Formulação com Múltiplas Origens de Conteúdos

As formulações apresentadas nas seções 5.1,5.2, 5.3 e 5.4 não tratam do caso em que existem múltiplas origens para os conteúdos. Uma possível formulação para tratar este aspectos é uma adaptação da formulação apresentada em [8], proposta para uma versão dinâmica do Problema de Posicionamento de Réplicas sem restrições de QoS.

Para esta formulação as seguintes notações foram utilizadas:

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t.
- $y_{kjt} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está replicado no servidor } j \text{ no período } t. \\ 0 & \text{caso contrário} \end{cases}$

- b_{it} *backlog* da requisição i no período t.
- $w_{kjlt} = \begin{cases} 1 & \text{se o conteúdo } k \text{ é copiado pelo servidor } j \text{ do servidor } l \text{ no período } t. \\ 0 & \text{caso contrário} \end{cases}$

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- T conjunto de períodos de tempo.
- L_k o tamanho do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- D_{it} demanda da requisição i no período t.
- $G(i)$ o conteúdo exigido pela requisição i.
- c_{ijt} custo de atendimento da requisição i no servidor j, no período t, calculado pela seguinte equação $c_{ijt} = (RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times BR_i$.
- p_{it} penalidade por usar *backlog* da requisição i no período t.

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} + \sum_{i\in R}\sum_{t\in T} p_{it}b_{it} + \sum_{k\in C}\sum_{j\in S}\sum_{l\in S}\sum_{t\in T} L_k w_{kjlt} \tag{5.44}$$

S.a.

$$\sum_{j\in S} L_{G(i)}x_{ijt} - b_{i(t-1)} + b_{it} = D_{it},\ \forall i \in R, \forall t \in T, \tag{5.45}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq MB_j, \forall j \in S, \forall t \in T, \tag{5.46}$$

$$\sum_{j\in S}\sum_{t\in T} x_{ijt} = 1,\ \forall i \in R, \tag{5.47}$$

$$y_{G(i)jt} \geq x_{ijt},\ \forall i \in R,\ \forall j \in S, \forall t \in T, \tag{5.48}$$

$$y_{kj(t+1)} - y_{kjt} \leq \sum_{l\in S} w_{kjlt}, \forall k \in C, \forall j \in S, \forall t \in T, \tag{5.49}$$

$$y_{kjt} \geq w_{kljt}, \forall k \in C, \forall j, l \in S, \forall t \in T, \tag{5.50}$$

$$\sum_{k\in C} L_k\ y_{kjt}\ \leq\ AS_j, \forall j \in S, \forall t \in T, \tag{5.51}$$

$$x_{ijt} \in [0,1], \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.52}$$

$$y_{kjt} \in \{0,1\}, \forall j \in S, \forall k \in C, \forall t \in T, \tag{5.53}$$

$$b_{it} \geq 0, \forall i \in R, \forall t \in T, \tag{5.54}$$

$$w_{kjlt} \in \{0,1\}, \forall j, l \in S, \forall k \in C, \forall t \in T. \tag{5.55}$$

As função objetivo, exposta em (5.44), 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. As restrições (5.45) associam as variáveis x_{ijt} e b_{it}, dizendo que a soma das quantidades entregues no dia atual, mais a quantidade que se ficará devendo para o dia posterior é igual à demanda do dia atual mais o que se ficou devendo do dia anterior. As restrições (5.46) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (5.47) exigem que uma requisição seja plenamente atendida. As restrições (5.48) condicionam que uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (5.49) garantem que toda replicação crie uma nova réplica. As restrições (5.50) exigem que uma replicação ocorra a partir de um servidor que possua o conteúdo replicado. As restrições (5.51) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As demais restrições são as de integralidade e não negatividade.

Esta formulação é obtida através da junção de conceitos das formulações apresentadas

anteriormente neste capítulo e da formulação apresentada em [8]. Esta formulação trata a modelagem de QoS através da técnica de *backlog*, restrições de espaço disponível nos servidores e também do caso de servidores heterogêneos, características estas não presentes na formulação proposta em [8]. Outra diferença entre a formulação apresentada nesta Seção e a proposta em [8] é o tipo das variáveis de custo operacional. Em [8] estas variáveis são inteiras e na formulação apresentada nesta seção estas variáveis são contínuas. Esta diferença de tipos é decorrente das diferenças entre os problemas tratados por estas duas formulações.

A formulação apresentada nesta seção modela o problema de maneira mais natural pois foi concebida a partir de uma formulação feita para um problema de posicionamento de réplicas. Além disso, a formulação original, proposta em [8], trata de aspectos que ainda não tratados pelas formulações apresentadas nas seções anteriores, como por exemplo, o caso de atualização de conteúdos e a possibilidade de cada conteúdo possuir sua própria origem.

5.6 Formulação com Submissão e Remoção de Conteúdos

Para modelar a atualização de conteúdos, é proposta uma abordagem baseada em tempo de vida para os conteúdos como meio de incorporar este novo aspecto. Tendo por base esta estratégia, uma nova formulação é proposta com base na formulação apresentada na Seção 5.5. A nova formulação trata dos aspectos de múltiplas origens para os conteúdos e de atualização, aspectos estes que não são tratados simultaneamente pelas formulações anteriores.

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t.
- $y_{kjt} = \begin{cases} 1 & \text{se o conteúdo } k \text{ está replicado no servidor } j \text{ no período } t. \\ 0 & \text{caso contrário} \end{cases}$
- b_{it} *backlog* da requisição i no período t.
- $w_{kjlt} = \begin{cases} 1 & \text{se o conteúdo } k \text{ é copiado pelo servidor } j \text{ a partir do} \\ & \text{servidor } l \text{ no período } t. \\ 0 & \text{caso contrário} \end{cases}$

Constantes:

- R conjunto de requisições a serem atendidas.
- S conjunto de servidores da RDC.
- C conjunto de conteúdos replicados.
- T conjunto de períodos de tempo.
- δ duração do período em segundos.
- L_k o tamanho do conteúdo k.
- B_k período em que o conteúdo k é disponibilizado.
- E_k período em que o conteúdo k é removido da RDC
- O_k servidor origem do conteúdo k.
- AS_j espaço em disco disponível no servidor j.
- MB_j banda máxima do servidor j.
- D_{it} demanda da requisição i no período t.
- BR_i banda mínima exigida pela requisição i.
- BX_i banda máxima aceita pela requisição i.
- $G(i)$ o conteúdo exigido pela requisição i.
- c_{ijt} custo de atendimento da requisição i no servidor j, no período t, calculado pela seguinte equação $c_{ijt} = (RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times BR_i$.
- p_{it} penalidade por usar *backlog* da requisição i no período t calculada pela equação $p_{it} = max(c_{ijt}) \times 2$.
- h_{kjlt} custo para replicar o conteúdo k no servidor j a partir do servidor l no período t.

$$Min \quad \sum_{i\in R}\sum_{j\in S}\sum_{t\in T} c_{ijt}x_{ijt} + \sum_{i\in R}\sum_{t\in T} p_{it}b_{it} + \sum_{k\in C}\sum_{j\in S}\sum_{l\in S}\sum_{t\in T} h_{kjlt}w_{kjlt} \tag{5.56}$$

S.t.

$$\sum_{j\in S} L_{G(i)}x_{ijt} - b_{i(t-1)} + b_{it} = \delta D_{it}, \forall i \in R, \forall t \in [B_{G(i)}, E_{G(i)}], \tag{5.57}$$

$$\sum_{i\in R} L_{G(i)}x_{ijt} \leq \delta M B_j, \ \forall j \in S, \forall t \in T, \tag{5.58}$$

$$\sum_{j\in S} L_{G(i)}x_{ijt} \leq \delta B X_i, \ \forall i \in R, \forall t \in T, \tag{5.59}$$

$$\sum_{j\in S}\sum_{t\in T} x_{ijt} = 1, \ \forall i \in R, \tag{5.60}$$

$$y_{G(i)jt} \geq x_{ijt}, \ \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.61}$$

$$\sum_{j\in S} y_{kjt} \geq 1, \ \forall k \in C, \forall t \in [B_k, E_k], \tag{5.62}$$

$$y_{kjt} = 0, \ \forall k \in C, \forall j \in S, \forall t \notin [B_k, E_k], \tag{5.63}$$

$$y_{kO_kB_k} = 1, \ \forall k \in C, \tag{5.64}$$

$$y_{kjB_k} = 0, \ \forall k \in C, \forall j \in \{S | j \neq O_k\}, \tag{5.65}$$

$$y_{kj(t+1)} \leq \sum_{l\in S} w_{kjlt}, \ \forall k \in C, \forall j \in S, \forall t \in T, \tag{5.66}$$

$$y_{kjt} \geq w_{kljt}, \ \forall k \in C, \forall j, l \in S, \forall t \in T, \tag{5.67}$$

$$\sum_{k\in C} L_k \ y_{kjt} \ \leq \ AS_j, \ \forall j \in S, \forall t \in T, \tag{5.68}$$

$$x_{ijt} \in [0,1], \ \forall i \in R, \forall j \in S, \forall t \in T, \tag{5.69}$$

$$y_{kjt} \in \{0,1\}, \ \forall j \in S, \forall k \in C, \forall t \in T, \tag{5.70}$$

$$b_{it} \geq 0, \ \forall i \in R, \forall t \in T, \tag{5.71}$$

$$w_{kjlt} \in \{0,1\}, \ \forall j, l \in S, \forall k \in C, \forall t \in T. \tag{5.72}$$

A função objetivo, exposta em (5.56), 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. As restrições (5.57) associam as variáveis x_{ijt} e b_{it}, dizendo que a soma das quantidades entregues no dia atual, mais a quantidade que se ficará devendo para o dia posterior é igual à demanda do dia atual mais o que se ficou devendo do dia anterior. As restrições (5.58) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (5.59) impedem que seja entregue ao cliente uma banda maior do que ele suporta. As restrições (5.60) exigem que uma requisição

seja plenamente atendida. As restrições (5.61) condicionam que uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (5.62) e (5.63) controlam 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 período de vida. As restrições (5.64) e (5.65) fazem com que no período de surgimento de um conteúdo apenas o servidor origem deste conteúdo possua uma réplica. As restrições (5.66) garantem que toda replicação crie uma nova réplica. As restrições (5.67) exigem que uma replicação ocorra a partir de um servidor que possua o conteúdo replicado. As restrições (5.68) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As demais restrições são as de integralidade e não negatividade.

Esta formulação, chamada a partir deste ponto de *Formulação Dinâmica* (*FD*), contempla todos os pontos abordados neste trabalho e é usada para obter limites de qualidade para comparações com as demais abordagens.

5.7 Um Algoritmo Enumerativo para o PPRDR

Testes computacionais revelaram que a formulação *FD*, apresentada na Seção 5.6, encontra dificuldades na resolução de algumas instâncias de pequeno porte (até 50 servidores) e praticamente é incapaz de resolver instâncias de grande porte (mais de 50 servidores). Nas instâncias de pequeno porte, a formulação FD encontra dificuldades nas instâncias em que os recursos (espaço em disco e banda nos servidores) são mais escassos e os canais de comunicação são assimétricos, ou seja, o tempo em que um pacote leva para ir de um servidor j para um segundo servidor l é diferente do tempo que um pacote leva para ir de l para j. Nas instâncias de grande porte o principal problema encontrado é o alta demanda de memória da formulação *FD*. Para instâncias de 50 servidores, a formulação *FD* produz modelos matemáticos que ocupam mais de 1 GB de memória, o que torna a resolução destes modelos uma tarefa difícil, visto que os algoritmos de resolução de problemas inteiros mistos tipicamente aumentam o tamanho do modelo matemático através da inclusão de cortes e ainda utilizam uma estrutura de árvore [53]. Esta grande demanda por memória dificulta o uso da formulação *FD* para as instâncias de grande porte do PPRDR ao ponto de não conseguir fornecer soluções inteiras para o problema e, em alguns casos, não ser capaz de resolver a relaxação linear. Devido a este fato, um algoritmo de enumeração para a resolução do PPRDR é proposto nesta seção com o objetivo de fornecer alguma base de comparação para as diversas heurísticas propostas,

uma vez que a formulação *FD* não é capaz de fornecer soluções inteiras para o PPRDR. Este algoritmo, chamado *Smart Enumeration Algorithm* (*SEA*), pode ser utilizado para obtenção de soluções ótimas e limites para as todas as instâncias do PPRDR, no entanto, optou-se por utilizá-lo apenas nas instâncias onde a formulação *FD* não é efetiva.

A estratégia do SEA é a mesma utilizada pelo algoritmo de *branch-and-bound* [53], que consiste em descrever o espaço de soluções através de uma árvore binária de decisões e explorar esta árvore para encontrar o ótimo global. Um ponto chave para este tipo de estratégia é como explorar de maneira eficiente a árvore. O algoritmo de *branch-and-bound* utiliza propriedades matemáticas dos problemas lineares para tornar mais eficiente o processo de exploração da árvore. A cada nó da árvore, a relaxação linear do problema tratado é resolvida e caso esta relaxação apresente um valor de função objetivo pior do que a melhor solução inteira encontrada até o momento, os filhos deste nó podem ser descartados uma vez que qualquer solução inteira encontrada a partir deste nó terá um valor de função objetivo igual ou pior que o valor da função objetivo da relaxação linear. Este mecanismo de descarte é conhecido como *poda* e é extremamente conhecido na literatura. O algoritmo proposto nesta seção também faz uso do mecanismo de poda porém, ao invés de usar a relaxação linear do problema, ele utiliza propriedades do PPRDR para tentar evitar nós da árvore que levam a soluções inviáveis ou de baixa qualidade.

A árvore que descreve o espaço de soluções usado pelo SEA pode ser obtida através dos seguintes passos:

- 1) Criar tuplas $< t, s, c >$ onde $t \in T$, $s \in S$ e c pertence ao conjunto de conteúdos ativos em t (Conteúdos que possuem t em seu tempo de vida). Armazenar as tuplas em uma lista l. Estas tuplas serão os nós da árvore.
- 2) Ordene l em ordem ascendente de i) períodos, ii) servidores e iii) conteúdos.
- 3) Suponha que a notação $x : xs$ representa uma lista com nó cabeça x e cauda xs. Seja $f(ramo, x : xs)$ uma função recursiva que recebe um ramo e uma lista de nós, remove o elemento na cabeça da lista (x), liga este elemento em $ramo$ criando dois sub-ramos, chamados de $ramoS$ e $ramoN$, que representam, respectivamente, a inserção do conteúdo $x.c$ no servidor $x.s$ no período $x.t$ e a **não** inserção de $x.c$ em $x.s$ no período $x.t$. A função f então si invoca recursivamente para ambos os ramos, usando a lista xs como parâmetro ($f(ramoS, xs)$ and $f(ramoN, xs)$). A função f retorna quando a lista recebida como parâmetro é vazia.

Para construir a árvore de decisões para o PPRDR, é necessário criar um nó r que é

a raiz da arvore e invocar a função f definida no passo 3 usando a lista l, construída no passo 2, e o nó r como parâmetros $(f(r, l))$.

A árvore construída pelos passos acima é binária e completa com altura dada por $|T| \times |S| \times |R|$ e $2^{|T|\times|S|\times|R|}$ folhas. Uma solução completa para o posicionamento de réplicas do PPRDR é obtida percorrendo todo o caminho entre a raiz r e uma folha, o que significa que para atingir uma solução completa para o posicionamento de réplicas, $|T| \times |S| \times |R|$ decisões binárias precisam ser tomadas. A cada nó, exitem apenas dois possíveis caminhos para atingir uma folha: $ramoS$ e $ramoN$. Caso $ramoS$ seja escolhido, para um nó $< p, s, c >$, significa que o conteúdo c está posicionado no servidor s no período t. Caso o caminho escolhido seja $ramoN$, isto significa que o conteúdo c **não** está posicionado no servidor s no período t.

Esta árvore pode se tronar demasiadamente grande mesmo para instâncias de pequeno porte, e por esta razão, construir toda a árvore não é razoável. Para explorar a estrutura de um modo mais eficiente, optou-se por construir a árvore sob demanda, o que significa que apenas um ramo é utilizado por vez. Deste modo, avaliando os ramos, um de cada vez, é possível encontrar as soluções ótimas para o PPRDR sem que a memória seja consumida em demasia. A ordenação das tuplas na lista l, faz com as decisões de um período t qualquer, só sejam tomadas após as decisões dos períodos anteriores à t. Isto possibilita que, depois de tomar todas as decisões de um período t qualquer, o PDR correspondente a t possa ser resolvido tendo como entrada, as decisões estabelecidas na árvore, as requisições ativas no período t e o *backlog* do período $t - 1$. Por requisições ativas em um período t entenda-se as requisições que chegam no período t e as requisições que chegam em períodos anteriores e que ainda não foram completamente atendidas.

Apesar de uma busca ramo a ramo na árvore ser perfeitamente possível, pode levar muito tempo para explorar toda a estrutura. Devido a isto, alguns mecanismos de poda são propostos com o objetivo de evitar a exploração de toda a estrutura da árvore e assim reduzir o tempo computacional necessário para encontrar as soluções ótimas.

- *Poda por violação do espaço em disco*: Ao analisar os conteúdos que estão posicionados em um mesmo servidor em um dado período, é possível determinar se a restrição de espaço em disco no servidor está ou não violada. Assim, se em algum nó da árvore esta violação for encontrada, o ramo contendo este nó, e consequentemente todos os seus filhos, pode ser descartado na busca uma vez que nenhuma solução viável pode ser encontrada a partir deste ramo.

- *Poda por conteúdo ausente*: Após tomar todas as decisões referentes a um único período, é possível determinar se todos os conteúdos ativos possuem pelo menos uma réplica. No caso afirmativo, o algoritmo de busca pode prosseguir para as decisões do próximo período. Caso contrário, o algoritmo pode retroceder na árvore e o ramo que apresentou a falta de um conteúdo pode ser descartado pelo fato de que nenhuma solução viável pode ser gerada a partir deste ramo.

- *Poda por falta de espaço em disco*: Ao analisar todos os conteúdos alocados no mesmo período é possível descartar ramos baseado na soma dos tamanhos dos conteúdos. Em qualquer nó, é possível calcular o valor x que representa a soma dos tamanhos dos conteúdos ausentes (conteúdos sem nenhuma réplica) no período. Também é possível calcular o valor y que representa a soma do espaço em disco disponível nos servidores. Caso o valor de x seja maior que o valor de y, o ramo contendo esta violação pode ser descartado pois não há possibilidade de encontrar soluções viáveis neste ramo.

- *Poda por custo mínimo adiante*: No PPRDR, requisições dos clientes normalmente levam mais de um período para ser completamente atendidas e é possível determinar o número mínimo de períodos necessários para atender cada requisição. Além disso, é possível calcular o custo mínimo de atendimento para cada requisição. Ao se computar a soma dos custos mínimos para todas as requisições ativas em um período é possível determinar um limite inferior para o custo de atendimento das requisições neste período. Baseado no limite para todos os períodos é possível podar ramos que levarão à soluções de qualidade inferior à melhor solução encontrada até o momento. Para tanto, é necessário manter um registro da melhor solução encontrada até o momento no processo de busca. Quando todas as decisões de um mesmo período forem tomadas é possível calcular a função objetivo da solução parcial construída até este período. Em qualquer nó intermediário n de um ramo da árvore é possível calcular um valor x que é a soma dos limites inferiores para o custo de atendimento das requisições de todos os períodos adiante de $n.p$. Caso seja constatado que a função objetivo da solução parcial construída até nó n acrescida de x é pior que o valor da função objetivo da melhor solução encontrada até o momento, não há necessidade de explorar mais este ramo uma vez que nenhuma solução fornecida por ele é um ótimo global.

Juntamente com os mecanismos de poda, uma heurística para gerar uma solução inicial também pode ser utilizada. Os mecanismos de poda comparam a solução do ramo

atual com a melhor solução encontrada até o momento, mas não há restrições a respeito do método usado para gerar a primeira melhor solução. Como o SEA tenta não explorar ramos que levam a soluções piores que a solução incumbente, uma boa heurística (ou outro método) pode ser usado uma solução inicial de alta qualidade, acelerando assim o procedimento de busca uma vez que, quanto melhor for a qualidade da primeira melhor solução menor será o número de ramos explorados pelo algoritmo.

O Algoritmo 1 apresenta um possível pseudo-código para o SEA. O algoritmo procede da seguinte maneira: Primeiramente, uma solução inicial é gerada por uma heurística ou outro método qualquer para ser usada como primeira melhor solução. A seguir, a lista de tuplas contendo as variáveis de decisão é criada e ordenada convenientemente. O passo mais importante do algoritmo é executado pela função *ExplorarRamo* que de fato executa o processo de enumeração e poda. A melhor solução encontrada pelo método *ExplorarRamo* é retornada como melhor solução encontrada pelo SEA.

Algoritmo 1 SEA()

1: Solução melhor = gerarSoluçãoInicial()
2: Construir a lista de tuplas l composta por $< t, s, c >, \forall t \in T, \forall s \in S$ e $\forall c \in$ *conteúdosAtivos(t)*
3: Ordenar l por i) período, ii) servidor e iii) conteúdo
4: Solução atual = novaSolução()
5: ExplorarRamo(atual, melhor, l)
6: Retorne melhor

O método *ExplorarRamo* é descrito pelo Algoritmo 2 e funciona da seguinte maneira: Primeiramente, é verificada a existência de alguma violação no espaço em disco das soluções (linhas 1–4). Caso esta violação seja encontrada não há necessidade de percorrer o ramo corrente e o método retorna para que outro ramo seja explorado. O segundo passo é a verificar se existe espaço em disco suficiente para alocar todos os conteúdos ausentes (linhas 5–8). Caso não exista espaço suficiente não há necessidade de explorar mais o ramo corrente e algoritmo retorna para que outro ramo seja explorado. O terceiro passo (linhas 9–21) é feito somente quando todas as decisões para um determinado período foram tomadas. Caso existam conteúdos ausentes, o algoritmo retorna devido ao fato de que nenhuma solução viável será encontrada neste ramo. Caso não existam inviabilidades na solução encontrada até o momento, verifica-se qualidade desta solução (linhas 14–20), podendo o ramo ser descartado caso seja constatado que ele não pode gerar uma solução melhor que a solução incumbente. O quarto passo (linhas 22–28) é feito somente quando todas as decisões foram tomadas. Neste caso, verifica-se se a solução produzida é melhor que a solução incumbente, sendo esta última atualizada se necessário. O quinto passo é o

passo recursivo do algoritmo (linhas 29–34).

Apesar da versão do SEA apresentada nos algoritmos 1 e 2 ser uma versão recursiva, o algoritmo implementado e usado nos testes computacionais é uma versão iterativa, que faz uso de estruturas de pilhas para substituir a recursão. Como a versão iterativa do algoritmo é muito mais complexa e extensa, por simplicidade, optou-se por expor a versão recursiva do *SEA*, que expressa os mesmos conceitos da versão iterativa de maneira mais simples e elegante.

Um fato importante a ser mencionado é que o *SEA* pode ser adaptado para armazenar todas as soluções ótimas encontradas durante sua execução. Para isto, basta fazer com que uma solução s seja armazenada em um conjunto de soluções no caso em que a função objetivo de s for igual à função objetivo da melhor solução encontrada até então. Caso uma solução melhor do que as que estão presentes no conjunto de soluções seja encontrada, significando que as soluções do conjunto são na verdade ótimos locais, todas as soluções do conjunto devem ser descartadas e um novo conjunto contendo apenas a nova melhor solução deve ser criado.

Como o tempo computacional deste algoritmo pode ser muito longo, um limite de máximo de tempo também é usado como critério de parada. Assim, o algoritmo *SEA* para quando termina de explorar toda a árvore ou quando sua execução atinge um tempo maior do que o limite. No primeiro caso, a solução encontrada é um ótimo global para o problema, no segundo, a solução retornada é a melhor solução encontrada no processo de busca.

Algoritmo 2 ExplorarRamo(Solução *sol*, Solução *melhor* , lista de tuplas l)

```
1: se Existe alguma violação da restrição de disco em sol então
2:    Poda por violação de espaço em disco
3:    Retorne
4: fim se
5: se Não há espaço para acomodar todos os conteúdos do período corrente em sol então
6:    Poda por falta de espaço em disco
7:    Retorne
8: fim se
9: se Todas as decisões de um período t foram tomadas então
10:   se Existe algum conteúdo ausente no período t em sol então
11:      Poda por conteúdo ausente
12:      Retorne
13:   fim se
14:   Seja c1 o valor da função objetivo do PDR correspondente ao período t em sol
15:   Seja c2 a soma dos valores das funções objetivos de todos os períodos anteriores a
      t em sol
16:   Seja c3 a soma dos custos mínimos de atendimento para todos os períodos posteri-
      ores a t
17:   se c1 + c2 + c3 é pior que a função objetivo de melhor então
18:      Poda por custo mínimo adiante
19:      Retorne
20:   fim se
21: fim se
22: se l está vazia então
23:   Todas as decisões foram tomadas
24:   se sol é melhor que melhor então
25:      melhor = sol
26:      Retorne
27:   fim se
28: fim se
29: Tupla q = removerCabeça(l)
30: ConfigurarAlocação(sol , q.t, q.s, q.c, 1)
31: ExplorarRamo(sol, melhor, l)
32: ConfigurarAlocação(sol , q.t, q.s, q.c, 0)
33: ExplorarRamo(sol, melhor, l)
34: inserirCabeça(q, l)
35: Retorne
```

Capítulo 6

Abordagens Heurísticas

Uma vez que abordagens exatas tendem a ser inviáveis à medida que as dimensões do dos problemas aumentam, e também devido as suas limitações em tratar modelos *online* [8], é natural que heurísticas sejam elaboradas para abordar instâncias e modelos em que os métodos exatos usualmente não apresentam bons resultados.

Neste capítulo são apresentados um *framework* unificador para heurísticas construtivas que particiona o PPRDR em dois subproblemas e combina as soluções destes subproblemas para formar uma solução para o PPRDR. Após a introdução do *framework*, são apresentadas técnicas para resolver o Problema de Distribuição de Requisições (PDR), que é um dos subproblemas tratados pelo *framework*. Após a apresentação das abordagens para o PDR, são apresentadas algumas técnicas de estimação de demanda que são usadas na resolução do Problema de Posicionamento de Réplicas (PPR) que é o segundo subproblema resolvido dentro do *framework* bem como algumas técnicas para resolver PPR. Após a exposição das técnicas para tratar cada um dos subproblemas, é apresentada uma coleção de heurísticas que podem ser descritas através do *framework* proposto como combinações das técnicas de resolução de cada subproblema. Por último, são mostradas abordagens adicionais que, apesar de não seguirem o *framework* proposto, possuem também sua importância na resolução do problema tratado.

6.1 *Framework* Unificador

Como consequência de um longo período dedicado a estudos de casos do problema abordado [36, 37, 38, 39], pode-se constatar que o PPRDR pode ser resolvido com uma margem de erro aceitável usando a técnica de divisão e conquista. O *framework* proposto parte do princípio que o PPRDR pode ser particionado em dois subproblemas: o Pro-

blema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR). A estratégia consiste em resolver cada um destes subproblemas separadamente, para no final fazer a junção das soluções dos dois subproblemas, transformando assim, esta junção de soluções para os dois subproblemas em uma solução para o PPRDR. Assim, um *framework* unificador para heurísticas construtivas para o PPRDR pode ser descrito pelo Algoritmo 3.

Algoritmo 3 *framework* para resolução do PPRDR()

```
1: para todo período de tempo faça
2:     Resolver o PDR
3:     Estimar demanda (Não Obrigatório)
4:     Resolver PPR
5: fim para
```

Este *framework* resolve o problema para cada período de tempo separadamente. Assim, para cada período, deve-se resolver o PDR usando a solução do PPR do período anterior. No caso do período inicial o posicionamento inicial dos conteúdos pode ser utilizado como entrada. Algumas abordagens para resolver o PDR, tanto exatas como heurísticas, são discutidas mais adiante neste capítulo. Em seguida deve-se de algum modo estimar a demanda para o próximo período. Este passo não é obrigatório, porém, as heurísticas que fazem uso deste passo são notoriamente melhores que as que não fazem [36]. Algumas técnicas para realização deste passo são discutidas mais adiante neste capítulo. O último passo é resolver o PPR. Este passo consiste em reorganizar os conteúdos fornecendo entrada para a resolução do PDR do próximo período. Algoritmos, exatos e heurísticos, para resolução do PPR são discutidos mais adiante.

Também são discutidos neste capítulo diversas combinações possíveis das diferentes técnicas para resolução de cada subproblema usadas para formar um algoritmo final para resolução do PPRDR.

6.2 Abordagens para o PDR

Esta seção apresenta técnicas exatas e heurísticas para a resolução do PDR. Estas técnicas podem ser permutadas e combinadas com diferentes técnicas de previsão de demanda e resolução do PPR para formar novos e melhores algoritmos.

6.2.1 Formulação Matemática para o PDR

Observando a formulação *FD*, apresentada no Capítulo 5, pode-se notar claramente que esta formulação trata de um problema inteiro misto, ou seja, no qual existem variáveis contínuas e variáveis inteiras. Ao separar estes dois conjuntos de variáveis, percebe-se que as variáveis contínuas estão relacionadas à associação de requisições a servidores (PDR) e que as variáveis inteiras estão relacionadas com a replicação de conteúdos em servidores (PPR). Assim sendo, dado um esquema de replicação, que mostra em quais servidores estarão as réplicas de cada um dos conteúdos, resolver o PDR torna-se um problema linear contínuo. Considerando os fatos apresentados uma formulação matemática, aqui chamada de *RF*, baseada na formulação *FD* é proposta em [38] como abordagem exata para a resolução do PDR e apresentada a seguir

Para a formulação *RF* as seguintes notações tiveram sua semântica alterada:

Variáveis:

- x_{ij} fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período atual.
- b_i backlog da requisição i para o período seguinte.

Constantes:

- B_i *backlog* anterior da requisição i.
- Y_{kj} indica se o conteúdo k está replicado no servidor j. 1 se verdadeiro e 0 caso contrário.
- D_i demanda da requisição i no período atual.
- c_{ij} custo de atendimento da requisição i no servidor j, calculado pela seguinte equação $c_{ij} = (RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times BR_i$.
- p_i penalidade por usar backlog da requisição i fazendo com que parte da demanda mínima seja atendida no período seguinte.

As demais notações que aparecem em trechos da formulação mantêm a semântica da formulação *FD*, apresentada no Capítulo 5.

$$Min \quad \sum_{i \in R} \sum_{j \in S} c_{ij} x_{ij} + \sum_{i \in R} p_i b_i \tag{6.1}$$

$S.a.$

$$\sum_{j \in S} L_{G(i)} x_{ij} + b_i = D_i + B_i, \tag{6.2}$$

$$\sum_{i \in R} L_{G(i)} x_{ij} \leq M B_j, \forall j \in S, \tag{6.3}$$

$$\sum_{j \in S} L_{G(i)} x_{ij} \leq B X_i, \forall i \in R, \tag{6.4}$$

$$x_{ij} \leq Y_{G(i)j}, \forall i \in R, \forall j \in S, \tag{6.5}$$

$$0 \leq x_{ij} \leq 1, \forall i \in R, \forall j \in S, \tag{6.6}$$

$$b_i \geq 0, \forall i \in R. \tag{6.7}$$

É importante mencionar que esta formulação para o subproblema é linear e contínua, ou seja, não possui variáveis inteiras. Este fato facilita muito a resolução do problema, visto que um dos maiores complicadores para a resolução de problemas lineares é justamente a presença de variáveis inteiras. Sem as variáveis inteiras, o subproblema pode ser resolvido em tempo polinomial de maneira eficiente por qualquer resolvedor gratuito como o GLPK [6], por exemplo.

Apesar de se tratar de um método exato para a resolução do PDR, esta formulação está incluída junto das abordagens heurísticas pelo fato de ser usada como parte de diversos algoritmos para resolução do PPRDR como é mostrado mais adiante neste capítulo. Como esta formulação é usada para resolver apenas uma parte do problema, ela deve ser usado juntamente com outras abordagens, exatas ou heurísticas, para compor heurísticas híbridas para PPRDR. Uma heurística é chamada de híbrida neste trabalho quando usa conceitos de métodos exatos e heurísticos ao mesmo tempo. Logo, qualquer algoritmo que use a formulação *RF* para resolver uma parte do problema é classificada como método (heurística) híbrido (a).

6.2.2 Atendimento Local

Esta técnica de atendimento é usada por soluções de mercado encontrada em provedores reais de RDC [29]. Esta técnica permite que uma requisição seja atendida apenas pelo servidor de origem, não permitindo portanto que a requisição seja encaminhada para

outros servidores e nem atendida por múltiplos servidores ao mesmo tempo. Esta abordagem produz resultados de baixa qualidade para cenários onde existem restrições de QoS [36], entretanto, não pode deixar de ser mencionada uma vez que é utilizada em RDC reais.

6.2.3 Encaminhamento Baseado em Caminho Mínimo

Esta técnica é usada na literatura como pode ser visto em [8, 13]. Esta técnica permite que as requisições sejam encaminhadas para outros servidores, fazendo com que os clientes possam ser atendidos não só pelo servidor de origem mas também por servidores remotos. Contudo, esta técnica não permite que as requisições sejam atendidas por múltiplos servidores ao mesmo tempo.

Ao chegar de uma requisição, o servidor que a recebe verifica se possui uma réplica do conteúdo exigido. Caso possua esta réplica e a banda deste mesmo servidor, que é o servidor de origem desta requisição, seja suficiente para atendê-la, esta requisição é atendida localmente. Caso contrário, a requisição é encaminhada para o servidor mais próximo do servidor de origem da requisição que possui recursos para atendê-la. Em ambientes com recursos ilimitados, não é necessário estabelecer uma ordem de prioridade para o atendimento de requisições uma vez que como os servidores possuem banda infinita, todas as requisições serão atendidas pelo servidor que proporcionar o menor custo. Entretanto, em ambientes onde os recursos são limitados, em que pode não haver recursos suficientes para atender todas as requisições, é necessário estabelecer prioridades. Em ambientes de redes computacionais, as requisições que chegam primeiro geralmente têm prioridade sobre as que chegam mais tarde [28], sendo esta a prioridade usada neste trabalho. Como as requisições que chegam no mesmo período não podem ser diferenciadas pelo tempo de chegada, assumi-se que as requisições de identificador menor chegam antes dar requisições com identificador maior.

Uma possível pseudo-código para esta estratégia é apresentado pelo Algoritmo 4, chamado de *SP*. O algoritmo procede da seguinte maneira: primeiramente verifica-se a possibilidade de atendimento das requisições pelo servidor origem (linhas 1-6 do Algoritmo 4). Em seguida, caso ainda existam requisições sem atendimento, o algoritmo escolhe a requisição de maior prioridade, determina o servidor mais próximo que possui recursos para atendê-la e faz o encaminhamento da requisição caso exista um servidor apto. Caso não exista um servidor, é feito o *backlog* da requisição. Para efeitos deste algoritmo, um servidor é considerado apto se possui uma réplica do conteúdo solicitado pela requisição

Algoritmo 4 Algoritmo SP

```
1: para Cada Requisição r faça
2:    Determinar o servidor j origem da requisição
3:    se j está apto para atender r então
4:       Atender r localmente
5:    fim se
6: fim para
7: enquanto Existir requisições não atendidas faça
8:    Escolher uma requisição r de maior prioridade
9:    Determinar o servidor j apto a atender r e que esteja mais próximo da origem(r)
10:   se  Não existir tal servidor então
11:      Fazer backlog de r
12:   senão
13:      Atender r remotamente por j
14:   fim se
15: fim enquanto
```

e possui banda suficiente para atender o cliente.

6.2.4 Modelo de Fluxo em Rede

Esta Seção apresenta uma nova modelagem para o PDR baseada em conceitos matemáticos de fluxo em rede. A nova modelagem consiste em representar o PDR através de um grafo acíclico e direcionado. Para ilustrar o processo de construção do grafo, o seguinte exemplo será utilizado: Suponha uma instância para o PDR onde existem dois clientes denotados por $C1$ e $C2$ e que buscam pelos conteúdos 1 e 2 respectivamente. Suponha também que exitam dois servidores, denotados por $S1$ e $S2$ e que $S1$ possui apenas uma única réplica do conteúdo 1 e $S2$ possui uma única réplica do conteúdo 2. A seguir serão descritos os passos para construir a rede e também ilustrações para o exemplo.

- Criar um vértice para representar cada cliente. A Figura 6.1 mostra a criação destes vértices para o exemplo supracitado.
- Criar um vértice para representar cada servidor. A Figura 6.2 Ilustra o grafo com os vértices que representam os clientes e os servidores.
- Criar um vértice que represente um servidor de capacidade infinita, ou seja, um servidor de *backlog*. Este vértice será usado para modelar o atraso das requisições e sua construção pode ser vista na Figura 6.3.
- Criar outros dois vértices, sendo um para ser a origem do fluxo, que é chamado de

fs, e um vértice para ser o destino do fluxo, que é chamado de fd. O grafo com todos os vértices necessários para a transformação é exposto na Figura 6.4.

- Estabelecer as demandas nos vértices sendo esta igual a FQ em fd, $-FQ$ em fs e zero nos vértices intermediários. O valor de FQ representa a soma das demandas de todos os clientes e é dado por $\sum_{i \in R} D_i$, sendo R e D_i descritos na Seção 5.6.

- Criar arcos que ligam o vértice fs a cada um dos cliente. Para cada cliente, é adicionado um arco com origem em fs tendo este arco custo igual a zero e capacidade BX_i. Os referidos arcos serão associados a uma classe chamada a. A construção destes arcos pode ser apreciada pela Figura 6.5.

- Criar arcos ligando os clientes aos servidores de modo que estes arcos modelem o atendimento viável, ou seja, para cada cliente são criados arcos entre este cliente e os servidores que possuem uma réplica do conteúdo requisitado por este cliente. Estes arcos pertencem à classe b e possuem custo igual a $c_{ij}/L_{G(i)}$, onde c_{ij} é calculado como apresentado na Seção 5.6 e $L_{G(i)}$ é o tamanho do conteúdo solicitado pelo cliente i. A capacidade máxima de um arco (i,j) pertencente a esta classe é igual à banda máxima do cliente BX_i. A Figura 6.6 representa o grafo com todos os vértices e arcos criados até este passo.

- Criar arcos ligando os clientes ao servidor de *backlog*. Estes arcos pertencem à classe d e possuem o custo calculado da mesma forma que p_i (ver Seção 5.6), ou seja, possuem o mesmo valor que a penalidade por *backlog*. A capacidade destes arcos é infinita, visto que não há limites para a quantidade de *backlog*. A criação destes arcos pode ser vista na Figura 6.7.

- Criar arcos entre os clientes e os servidores de modo que estes arcos representem o atendimento inviável, ou seja, para cada cliente são criados arcos ligando este cliente a todos os servidores que não possuem uma réplica do conteúdo requisitado por este cliente. Estes arcos são criados somente para que o grafo não apresente vértices inalcançáveis, uma vez que um servidor sem réplicas não pode ser apontado por nenhum arco da classe b. Os arcos criados neste passo são associados à classe c e possuem capacidade igual à banda máxima do cliente BX_i. Como os arcos desta classe representam o atendimento inviável, seu custo deve ser muito maior que qualquer arco da classe d. A Figura 6.8 ilustra a criação destes arcos.

- Criar arcos ligando os servidores ao vértice fd. Para cada servidor, incluindo o servidor de *backlog*, é adicionado um arco, associado à classe e, que liga este servidor

ao vértice fd, com custo zero e capacidade igual à banda máxima do servidor. O grafo final da transformação pode ser visto na Figura 6.9.

A solução para o PDR é dada pela resolução do Problema de Fluxo de Custo Mínimo (PFCM)[7] na rede construída a partir da transformação descrita anteriormente.

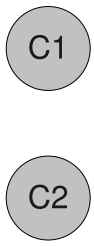

Figura 6.1: Modelagem em Rede para o PDR Passo 1

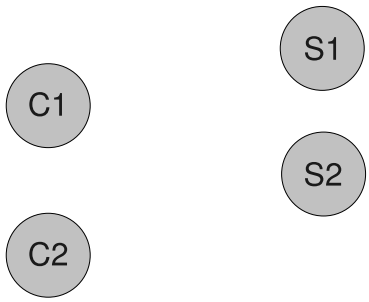

Figura 6.2: Modelagem em Rede para o PDR Passo 2

Teste preliminares com uma formulação matemática construída a partir desta modelagem revelam que ela não é capaz de tratar todos os aspectos da versão do PDR tratado neste trabalho. Em particular, a modelagem descrita não contempla os casos em que as requisições possuem uma demanda maior que a capacidade do cliente. Assim sendo, esta modelagem trata o backlog das requisições que é feito no período atual, mas não trata o backlog feito nos períodos anteriores.

Para resolver este problema uma segunda modelagem de fluxo em redes é proposta. Esta segunda modelagem é baseada na primeira, e para construir o grafo no qual o PFCM será resolvido os seguintes passos devem ser feitos:

- Criar um par de vértices para cada cliente. Deste par, um vértice será associado a um conjunto R' e o outro será associado a um conjunto R.

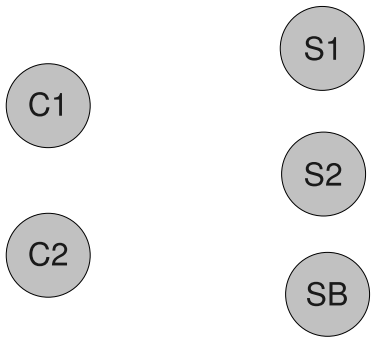

Figura 6.3: Modelagem em Rede para o PDR Passo 3

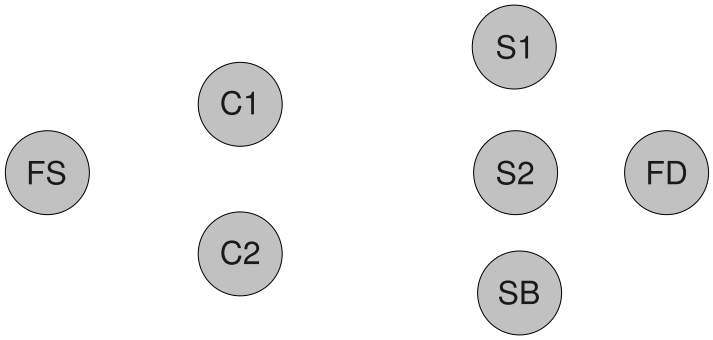

Figura 6.4: Modelagem em Rede para o PDR Passo 4

- Criar um vértice para cada servidor, assim como na modelagem anterior.
- Criar um vértice que represente um servidor de capacidade infinita, ou seja, um servidor de *backlog*.
- Criar outros dois vértices, sendo um para ser a origem do fluxo, que é chamado de fs, e um vértice para ser o destino do fluxo, que é chamado de fd.
- Estabelecer as demandas nos vértices sendo esta igual a FQ em fd, $-FQ$ em fs e zero nos vértices intermediários. O valor de FQ é dado por $\sum_{i \in R} D_i$, sendo R e D_i descritos na Seção 5.6.
- Criar arcos que ligam o vértice fs aos clientes. Para cada vértice de cliente em R', é adicionado um arco com origem em fs. Estes arcos pertencem à classe a, possuem custo igual a zero e capacidade D_i.
- Criar arcos entre os vértices que representam os Clientes. Cada cliente é representado por dois vértices sendo que um destes vértices pertence a R' e o outro pertence a R. Uma classe de arcos, chamada f, liga os vértices destes dois conjuntos, sendo

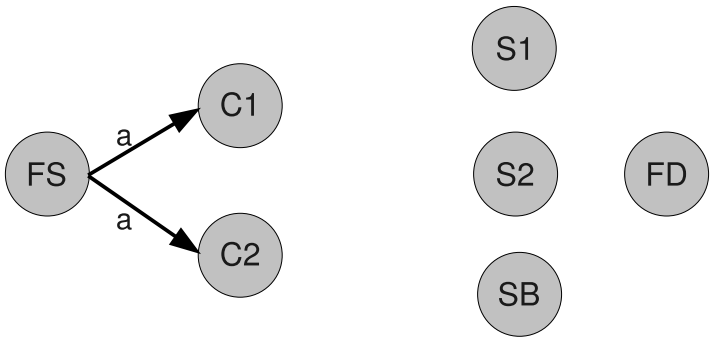

Figura 6.5: Modelagem em Rede para o PDR Passo 5

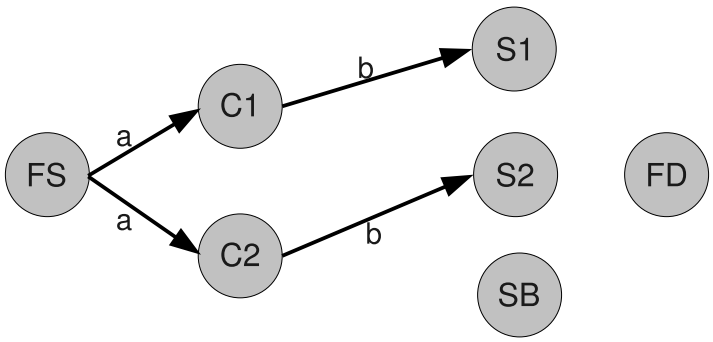

Figura 6.6: Modelagem em Rede para o PDR Passo 6

que para cada vértice $i \in R'$ é criado um arco da classe f que liga i à um vértice $j \in R$ sendo que o par de vértices $< i, j >$ representa o mesmo cliente.

- Criar arcos para ligar os vértices de R aos servidores de modo que estes arcos modelem o atendimento viável, ou seja, para cada vértice em $i \in R$ são criados arcos entre este vértice e os servidores que possuem uma réplica do conteúdo requisitado pelo cliente que i representa. Estes arcos pertencem à classe b e possuem custo igual a $c_{ij}/L_{G(i)}$, onde c_{ij} é calculado como apresentado na Seção 5.6 e $L_{G(i)}$ é o tamanho do conteúdo solicitado pelo cliente representado por i. A capacidade máxima de um arco (i, j) pertencente a esta classe é igual à banda máxima do cliente BX_i.

- Criar arcos ligando os vértices em R' ao servidor de *backlog*. Estes arcos pertencem à classe d e possuem o custo calculado da mesma forma que p_i (ver Seção 5.6), ou seja, possuem o mesmo valor que a penalidade por *backlog*. A capacidade destes arcos é infinita, visto que não há limites para a quantidade de *backlog*.

- Criar arcos entre os vértices de R e os servidores de modo que estes arcos representem o atendimento inviável, ou seja, para cada vértice em R são criados arcos ligando este

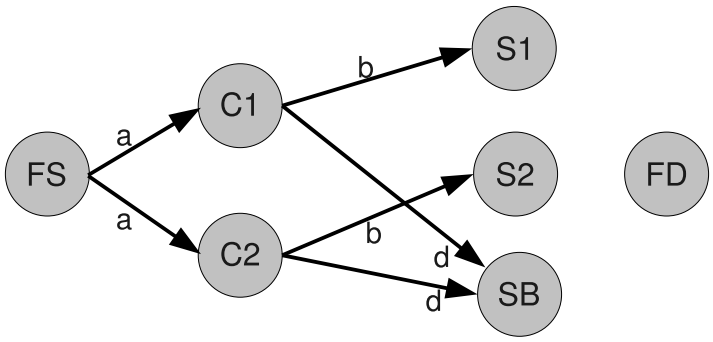

Figura 6.7: Modelagem em Rede para o PDR Passo 7

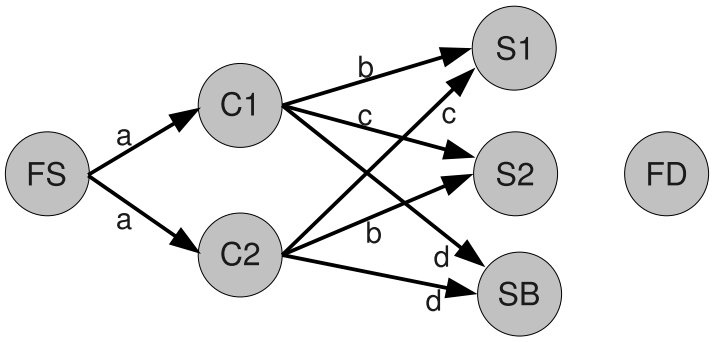

Figura 6.8: Modelagem em Rede para o PDR Passo 8

vértice a todos os servidores que não possuem uma réplica do conteúdo requisitado. Estes arcos são criados somente para que o grafo não apresente vértices inalcançáveis, uma vez que um servidor sem réplicas não pode ser apontado por um arco da classe b. Os arcos criados neste passo são associados à classe c e possuem capacidade igual à banda máxima do cliente BX_i. Assim como na modelagem anterior, pelo fato destes arcos representarem o atendimento inviável, o custo deles deve ser muito maior que qualquer arco da classe d.

- Criar arcos ligando os servidores ao vértice fd. Para cada servidor, incluindo o servidor de backlog, é criado um arco, associado à classe e, que liga este servidor ao vértice fd, com custo zero e capacidade igual à banda máxima do servidor.

A Figura 6.10 mostra a rede construída usando a segunda modelagem para o mesmo exemplo ilustrado anteriormente.

Esta Nova modelagem por fluxo em rede é capaz de modelar o backlog feito em períodos anteriores, visto que ela permite que um fluxo maior que a banda máxima do cliente chegue até os vértices que o representam. Como a quantidade de fluxo FQ é dada

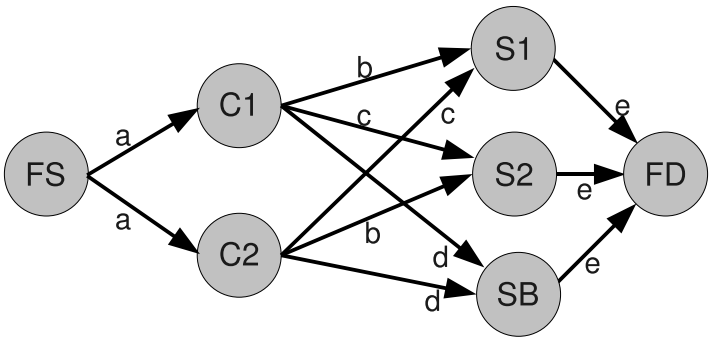

Figura 6.9: Modelagem em Rede para o PDR

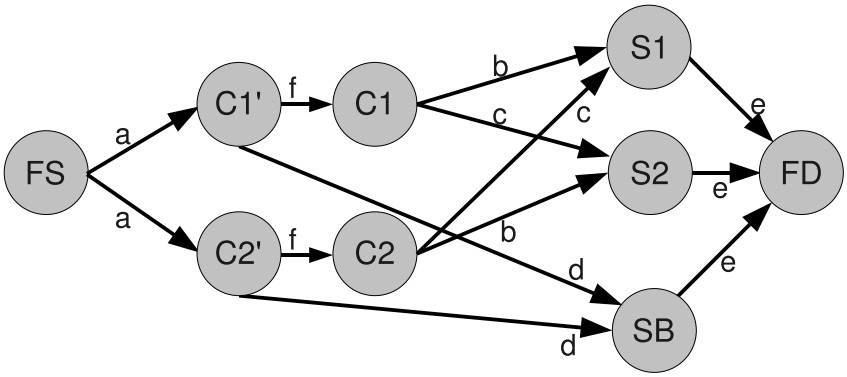

Figura 6.10: Modelagem em Rede para o PDR com Demanda Maior que a Capacidade

pela soma das demandas, e os arcos da classe a possuem capacidade igual à demanda de cada cliente, todos estes arcos serão saturados, significando que cada cliente recebe um fluxo equivalente à sua própria demanda. Como a capacidade máxima dos arcos da classe f, que ligam os vértices que representam um mesmo cliente, é igual à banda máxima do cliente, a demanda excedente é forçada a seguir pelos arcos da classe d, que representam o backlog das requisições. Isto significa que nenhum cliente tem mais do que sua capacidade de banda atendida. O fluxo que chega aos vértices em R deve escoar por arcos da classe b ou c. Caso não existam arcos da classe b, um algoritmo de fluxo de custo mínimo deve perceber que não é vantajoso escoar o fluxo por arcos da classe c, fazendo assim que quantidade de fluxo escoado por arcos da classe f seja transferido para os arcos da classe d. A capacidade de banda nos servidores é garantida pelos arcos da classe e, sendo portanto impossível que um servidor conduza mais fluxo que sua capacidade de banda. Caso a soma do fluxo que chega a um servidor seja maior que sua capacidade de saída, um algoritmo para o PFCM deve perceber que parte do fluxo está sendo retido nos nós intermediários, devendo portanto reconduzir este fluxo por outros arcos da classe b ou

ainda por arcos da classe d.

Esta modelagem foi utilizada para converter aproximadamente 1050 instâncias do PDR em instâncias do PFCM que foram resolvidas com sucesso utilizando a formulação descrita a seguir:

Variáveis:

- x_{ij} fluxo que passa no arco (i, j).

Constantes:

- A conjunto de arestas do grafo;
- V conjunto de vértices do grafo;
- R subconjunto de vértices que contém um dos vértices de cada clientes;
- R' conjunto de vértices que contém o segundo vértice de cada cliente;
- S conjunto de vértices que representam servidores;
- fs vértice fonte do fluxo;
- fd vértice destino do fluxo;
- c_{ij} custo de transporte por unidade do arco (i, j);
- q_{ij} capacidade do arco (i, j);
- FQ soma das demandas dos clientes.

$$Min \quad \sum_{i \in R} \sum_{j \in S} c_{ij} x_{ij} \tag{6.8}$$

$S.a.$

$$\sum_{j \in R'} x_{fs\,j} = FQ, \tag{6.9}$$

$$\sum_{i \in S} x_{i\,fd} = FQ, \tag{6.10}$$

$$\sum_{(j,i) \in A} x_{ji} = \sum_{(i,j) \in A} x_{ij}, \ \forall i \in V \setminus \{FS, FD\}, \tag{6.11}$$

$$x_{ij} \leq q_{ij}, \ \forall (i,j) \in A, \tag{6.12}$$

$$x_{ij} \in \mathbb{N}, \ \forall (i,j) \in A. \tag{6.13}$$

$$\tag{6.14}$$

Ao comparar os resultados obtidos para as instâncias do PFCM resolvidas através da formulação exposta por (6.8)-(6.13), com os resultados obtidos pela formulação *RF* para o PDR, pode ser constatado que as duas formulações apresentam o mesmo valor de função objetivo e que o valor final das variáveis de cada problema expressam o mesmo significado.

Estudos experimentais feitos com as duas formulações mostram que a formulação *RF* produz programas lineares em média 31.8% maiores que os produzidos pela formulação de fluxo em rede, no entanto, os tempos computacionais observados para resolução do problema usando a formulação de fluxo em rede são em média 87% maiores. Uma possível causa para este aumento nos tempos computacionais é que, apesar de produzir programas lineares menores, a formulação de fluxo em rede possui apenas variáveis inteiras, o que dificulta muito a resolução do problema via modelos matemáticos. Entretanto, o uso desta modelagem juntamente com um algoritmo adequado para resolvê-la pode produzir bons resultados em termos de tempo. Neste sentido, o algoritmo *Network Simplex* [7] é utilizado na resolução da modelagem proposta por ser um dos mais poderosos algoritmos de resolução para este tipo de modelagem e também por ter implementações extremamente eficientes disponíveis.

Esta abordagem por fluxo em rede comprova que o PDR, pode ser modelado como PFCM *single - commodity*, o que até então não havia sido feito na literatura. Esta nova modelagem também abre uma série de novas possibilidades para resolução do problema

através de abordagens baseadas em grafos.

6.2.4.1 Modelando Problemas de Transporte Multiproduto como Problemas de Fluxo de Custo Mínimo

Esta seção apresenta uma modelagem de fluxo em rede para uma versão do Problema de Transporte Multiproduto (PTM).

O sucesso no uso da metodologia descrita na Seção 6.2.4 para resolução do PDR relacionado a aproximadamente 1050 instâncias motiva um estudo para verificar se esta modelagem pode ser estendida para os demais problemas de transporte multiproduto.

O problema de transporte multiproduto pode ser definido da seguinte maneira: seja $N = \{c_1, c_2, ..., c_n\}$ o conjunto clientes, $M = \{f_1, f_2, ..., f_m\}$ o conjunto de facilidades e $K = \{p_1, p_2, ..., p_k\}$ o conjunto de produtos. Cada cliente i possui uma demanda específica por cada produto k que é denotada por d_{ik}. Cada facilidade j possui um limite de produção específico para cada produto k sendo denotado por l_{jk}. Um custo r_{ijk} é pago toda vez que uma unidade da demanda do cliente i pelo produto k for atendida pela facilidade j e ainda existe um limite máximo U_{ij} de unidades de produto que podem ser transportados da facilidade j para o cliente i. O objetivo do problema de transporte multiproduto é atender as demandas de todos os clientes por todos os produtos, pagando o menor custo possível e sem violar as capacidades das facilidades e dos arcos.

O PTM pode ser descrito pelo seguinte modelo matemático considerando x_{ijk} a quantidade de produtos do tipo k transportada da facilidade j para o cliente i.

$$Min \quad \sum_{k \in K} \sum_{i \in N} \sum_{j \in M} r_{ijk} x_{ijk} \tag{6.15}$$

$S.a.$

$$\sum_{i \in N} x_{ijk} \leq l_{jk}, \forall j \in M, \forall k \in K, \tag{6.16}$$

$$\sum_{j \in M} x_{ijk} \geq d_{ik}, \forall i \in N, \forall k \in K, \tag{6.17}$$

$$\sum_{k \in K} x_{ijk} \leq U_{ij}, \forall i \in N, \forall j \in M, \tag{6.18}$$

$$x_{ijk} \in \mathbb{N}, \ \forall i \in N, \forall j \in M, \forall k \in K. \tag{6.19}$$

A função objetivo mostrada em (6.15) minimiza os custos de transporte de produtos

entre facilidades e cliente. As restrições (6.16) asseguram que as capacidades de produção das facilidades para cada produto sejam respeitadas. As restrições (6.17) garantem que a demanda de cada cliente por cada produto seja atendida. As restrições (6.18) preservam a capacidade máxima de transporte entre as facilidades e os clientes e as restrições (6.19) são as restrições de integralidade e não negatividade.

As restrições (6.18) da formulação acima são de suma importância quando o limite que pode ser transportado entre uma facilidade e um cliente é menor que a capacidade de receptação do cliente. Um bom exemplo disso é o uso de óleo-dutos. Suponha uma refinaria de combustíveis ligada uma uma distribuidora através de um óleo-duto. A refinaria não pode enviar para a distribuidora um volume de combustível maior do que a capacidade do óleo-duto que liga estas duas facilidades. Contudo, quando se trata de redes de comunicação, a capacidade de transmissão normalmente é muito maior do que a capacidade de receptação dos clientes. Para estes casos, o PTM pode ser descrito através da seguinte formulação:

$$Min \quad \sum_{k\in K}\sum_{i\in N}\sum_{j\in M} r_{ijk}x_{ijk} \tag{6.20}$$

$S.a.$

$$\sum_{i\in N} x_{ijk} \leq l_{jk}, \forall j \in M, \forall k \in K, \tag{6.21}$$

$$\sum_{j\in M} x_{ijk} \geq d_{ik}, \forall i \in N, \forall k \in K, \tag{6.22}$$

$$\sum_{k\in K}\sum_{j\in M} x_{ijk} \leq q_i, \forall i \in N, \tag{6.23}$$

$$x_{ijk} \in \mathbb{N},\ \forall i \in N, \forall j \in M, \forall k \in K. \tag{6.24}$$

As restrições 6.23 são as únicas diferenças entre o modelo exposto por (6.15)-(6.19) e o modelo exposto por (6.20)-(6.24). Nestas restrições q_i representa o limite máximo de receptação do cliente i. Estas restrições garantem que o limite de receptação do cliente i seja respeitado independentemente de qual é o produto sendo transportado e da origem deste produto. Note que, $q_i \geq \sum_{k\in K} d_{ik}$ para todo cliente i em uma instância viável do problema. O modelo definido por (6.20)-(6.24) pode ser usado para resolver o PTM em casos onde o $\max(q_i)$ é menor que $\min(U_{ij})$ pois a limitação destes casos está na capacidade de receptação dos clientes e não na capacidade de transporte entre os clientes e facilidades.

Para tornar o problema mais geral, um novo conjunto de restrições para modelar a capacidade total de produção das facilidades é inserido. Estas restrições podem ser usadas em casos em que além de limites específicos para cada produto, uma facilidade possui um limite adicional modelando o número máximo de produtos que podem ser montados. Suponha que uma facilidade possua matéria prima para produzir, 2 unidades do produto A, 4 do produto B e 3 do produto C, porém só há funcionários disponíveis para produzir apenas 5 produtos. Assim sendo, qualquer combinação dos produtos A, B e C é válida desde que o número total de produtos não exceda 5.

O modelo matemático contendo este novo conjunto de restrições, onde o limite de produção total para cada facilidade é denotado por q_j é apresentado a seguir.

$$Min \quad \sum_{k \in K} \sum_{i \in N} \sum_{j \in M} r_{ijk} x_{ijk} \tag{6.25}$$

$S.a.$

$$\sum_{i \in N} x_{ijk} \leq l_{jk}, \forall j \in M, \forall k \in K, \tag{6.26}$$

$$\sum_{j \in M} x_{ijk} \geq d_{ik}, \forall i \in N, \forall k \in K, \tag{6.27}$$

$$\sum_{k \in K} \sum_{j \in M} x_{ijk} \leq q_i, \forall i \in N, \tag{6.28}$$

$$\sum_{k \in K} \sum_{i \in N} x_{ijk} \leq q_j, \forall j \in M, \tag{6.29}$$

$$x_{ijk} \in \mathbb{N}, \ \forall i \in N, \forall j \in M, \forall k \in K. \tag{6.30}$$

Neste modelo a função objetivo mostrada em (6.25) minimiza os custos de transporte de produtos entre facilidades e cliente. As restrições (6.26) asseguram que as capacidades de produção das facilidades para cada produto sejam respeitadas. As restrições (6.17) garantem que a demanda de cada cliente por cada produto seja atendida. As restrições (6.28) fazem com que as capacidades máximas de receptação dos clientes sejam respeitadas. As restrições (6.29) asseguram o limite máximo de produção das facilidades e as restrições (6.19) são as restrições de integralidade e não negatividade.

A Figura 6.11 mostra uma instância para o PTM no caso onde os limites de produção e receptação são menores que as capacidades de transporte. Nesta figura, os círculos de *c1* a *cn* representam clientes e os círculos de *f1* à *fm* representam facilidades. Acima de cada círculo existem k valores que representam as demandas pelos k produtos no caso

dos clientes e as capacidades de produção para os k produtos no caso das facilidades. A capacidade total de produção para cada facilidade é mostrada abaixo e à direita de cada facilidade.

Figura 6.11: Instância para o Problema de transporte Multiproduto

Para transformar uma instância deste PTM em uma instância do PFCM propõe-se a seguinte representação:

- Para cada cliente, devem ser criados $k + 1$ vértices, sendo um para representar o cliente e os demais vértices para modelar a demanda do cliente por cada produto. A Figura 6.12 ilustra a criação destes vértices quando são considerados n clientes e k produtos.

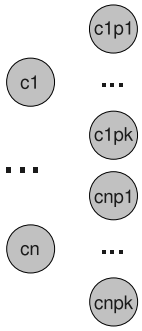

Figura 6.12: Passo 1: Representando os clientes e suas demandas

- Para cada facilidade devem ser criados $k + 1$ vértices, sendo um vértice para representar a facilidade e os demais para modelar a capacidade de produção da facilidade para cada produto. A Figura 6.13 representa um grafo com n clientes m facilidades e k produtos.

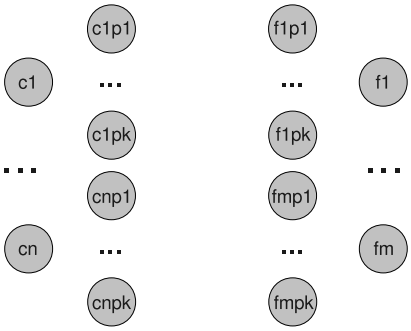

Figura 6.13: Passo 2: Representando os limites de produção de cada produto nas m facilidades

- Dois outros vértices artificiais devem ser criados, um para a origem do fluxo (sf) e um para o destino (df). A Figura 6.14 mostra o grafo após a criação destes vértices.

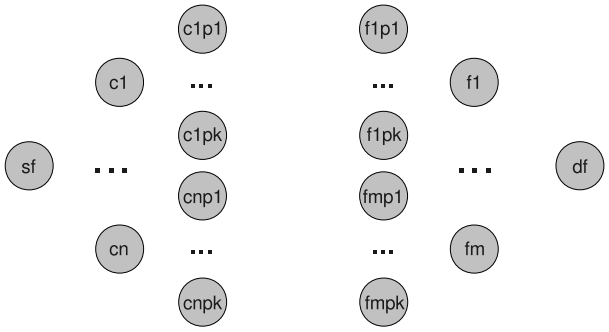

Figura 6.14: Passo 3: Criação dos vértices artificiais de origem e destino do fluxo

- Estabelecer as demandas de todos os vértices, sendo esta igual a $-FQ$ em sf, FQ em df e zero nos vértices intermediários, onde FQ é igual ao somatório das demandas de todos os clientes por todos os produtos, dada por $\sum_{i \in N} \sum_{k \in K} d_{ik}$.

- Criar arcos ligando o vértice sf a cada um dos clientes. Estes arcos possuem custo zero e capacidade igual a demanda total do cliente ao qual ele está ligado. A capacidade total de um cliente i pelos k produtos é dada por $\sum_{k \in K} d_{ik}$. A criação dos referidos arcos pode ser vista na Figura 6.15. Para este conjunto de arcos será criada uma classe representada por a.

- Criar arcos ligando cada cliente i a cada um dos k vértices que representam a demanda deste cliente. Cada arco (i,k) possui custo zero e capacidade igual d_{ik}.

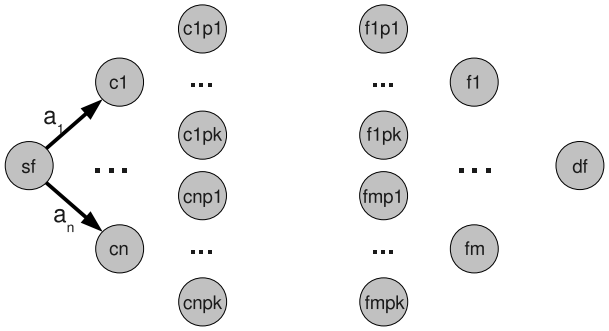

Figura 6.15: Passo 4: Representação dos arcos da classe a

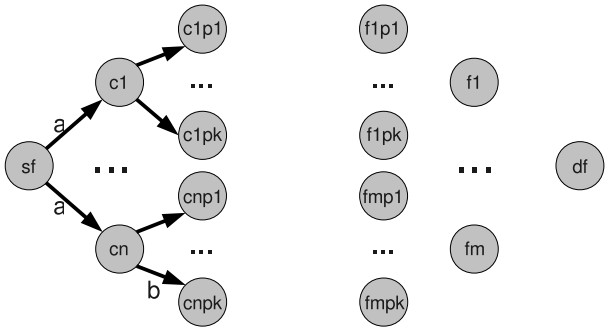

Figura 6.16: Passo 5: Representação dos arcos da classe b

A criação deste conjunto de arcos é mostrada na Figura 6.16. Uma nova classe, chamada de b, é criada para este conjunto de arcos.

- Para cada produto, criar arcos que ligam os vértices que representam as demandas deste produto aos vértices que representam as capacidades de produção deste produto. A capacidade destes arcos é denotada por d_{ik} e é equivalente a demanda do cliente pelo produto. Seu custo é representado por r_{ijk} e é igual ao custo de atendimento do cliente pela facilidade para o produto k. A Figura 6.17 ilustra a criação dos referidos arcos que serão associados à classe c. Note que os arcos desta classe ligam somente vértices que representam o mesmo produto. Assim sendo, não podem haver arcos que ligam vértices que representam demanda pelo produto p_i aos vértices que representam a capacidade de produção do produto p_j.

- Uma facilidade j tem capacidades de produção específicas para cada produto p_k, $p_k \in K$. Logo, deve-se criar arcos que ligam os vértices que representam a capacidade de produção dos k produtos ao vértice que representa a facilidade j. A capacidade

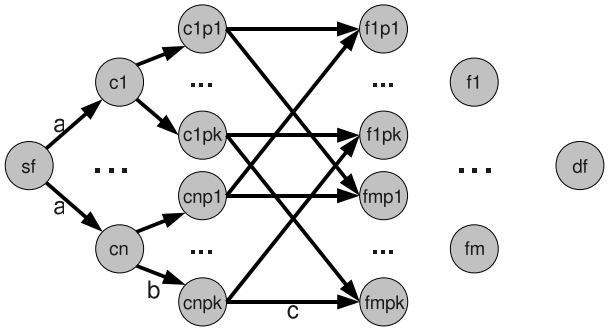

Figura 6.17: Passo 6: Representação dos arcos da classe c

destes arcos é denotada por l_{jk} e é igual à capacidade de produção da facilidade j para o produto k. A Figura 6.18 ilustra estes arcos que serão associados à classe d. Os arcos desta classe possuem custo igual à zero e representam o quanto a facilidade atendeu de cada produto.

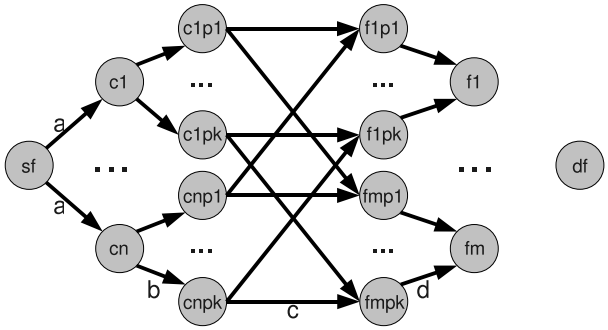

Figura 6.18: Passo 7: Representação dos arcos da classe d

- Criar arcos para ligar os vértices que representam as facilidades ao vértice df. Estes arcos possuem custo igual a zero e capacidade igual a capacidade total da facilidade q_j que é origem do arco. Estes arcos representam a quantidade total de produtos atendida pelas facilidades e serão associados à classe e.

Após a execução de todos os passos a solução do PTM é equivalente à solução do PFCM no grafo resultante das transformações propostas, anteriormente descritas, assim como mostra a Figura 6.19.

Assim como pode ser Observado na Figura 6.19, o número de vértices criados para montar o referido grafo é $(k+1) \times n + (k+1) \times m + 2$ sendo portanto da ordem de $O(k(n+m))$, onde k é a quantidade de produtos, n é o número de clientes e m o número

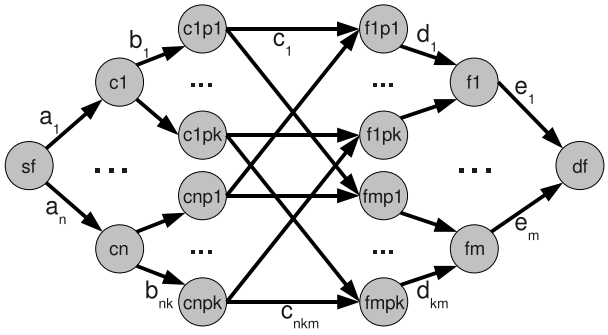

Figura 6.19: Passo 8: Representação dos arcos da classe e

	c1	c2	f1	f2
p1	0	2	-3	0
p2	2	2	-2	-4
p3	3	2	-4	-1

Tabela 6.1: Instância para o PTM

de facilidades. O número de arcos criados para montar o grafo é igual a n arcos para a classe a, $n \times k$ para a classe b, $n \times k \times m$ para a classe c, $k \times m$ para a classe d e m para a classe e. Logo, o número total de arcos do grafo é $n + n \times k + n \times k \times m + k \times m + m$, sendo da ordem de O($nkm$).

A Tabela 6.2.4.1 apresenta exemplo numérico para o PTM. Esta instância possui dois clientes, representados por c1 e c2, duas facilidades, representadas por f1 e f2, e três produtos, representados por p1, p2 e p3. O cliente c1 possui demanda de 0, 2 e 3 unidades pelos produtos 1, 2 e 3 respectivamente. O cliente c2 possui demandas de 2 unidades para todos os produtos. A facilidade 1 possui capacidade de produção 2,3 e 4 para os produtos 1,2 e 3 respectivamente. A facilidade 2 possui limites de produção iguais a 0 unidades para o produto 1, 4 unidades para o produto 2 e 1 unidade para o produto 3. O limite geral de produção q_j para ambas as facilidades é infinito. O custo de atendimento das demandas do cliente c1 para cada produto, a partir da facilidade f1 são dados pela expressão $r_{111} = r_{112} = r_{113} = 1$. Já os custos de atendimento das demandas do cliente c1 pela facilidade f2 são dados por $r_{121} = r_{122} = r_{123} = \sqrt{2}$. Os custos de atendimento das demandas do cliente c2 pelas facilidades f1 e f2 são dados por $r_{211} = r_{212} = r_{213} = \sqrt{2}$ e $r_{221} = r_{222} = r_{223} = 1$ respectivamente.

A modelagem de fluxo em rede para a instância da Tabela 6.2.4.1 é mostrada na figura 6.20. A partir das considerações realizadas sobre a Figura 6.19, pode-se notar que naquela figura a criação de 18 vértices e 28 arcos sendo 2 arcos para a classe a, 6 para a classe b, 12 para a classe c, 6 para classe d e 2 para a classe e. Para exemplificar os

custos e capacidades dos arcos, os valores de um único arco pertencente à cada classe são mostrados a seguir. Na classe a, por exemplo, o arco a_1 possui custo zero e capacidade dada por $\sum_{k \in K} d_{1k} = 5$. Na classe b o arco b_6 tem custo zero e capacidade dada por $d_{23} = 2$. Na classe c o arco c_{12} tem capacidade dada por $d_{23} = 2$ e custo dado por $r_{223} = 1$. Na classe d o arco d_1 tem custo zero e capacidade da por $l_{11} = 3$ e na classe e o arco e_1 tem custo zero e capacidade infinita. A demanda por fluxo no vértice df é dada por $FQ = \sum_{i \in \{1,2\}} \sum_{k \in \{1,2,3\}} d_{ik} = 11$, logo, a demanda em sf é -11 e 0 nos demais vértices.

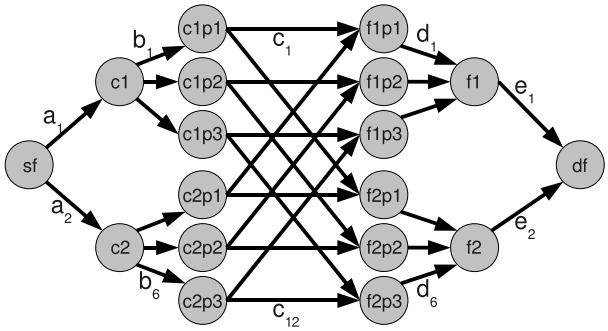

Figura 6.20: Modelagem em Rede para o Problema de transporte multiproduto

Como pode ser observado, a complexidade total da transformação proposta é da ordem de O($k(n+m)$)+O(nkm) =O(nkm), sendo portanto, executada em um número polinomial de passos. Como o PFCM pode ser resolvido em tempo polinomial[7], obtém-se então, um algoritmo polinomial para a resolução do PTM nos casos onde: 1) existe um limite máximo de receptação para os clientes, 2) não existem limites máximos de transporte entre clientes e facilidades e 3) exitem limites máximos de produção nas facilidades. Esta modelagem para o PTM foi concebida após a estudos feitos com a modelagem específica para o PDR e o estudo de um exemplo apresentado em [7], logo esta modelagem é fortemente baseada [7], contudo, a modelagem apresentada em [7] não apresenta limites totais de produção nas facilidades e não apresenta uma análise de complexidade. O modelo matemático do problema e sua relação com o PTM tradicional também não são expostos em [7], fatos estes que, no entendimento do autor, justificam a inclusão destes tópicos nesta seção.

6.2.5 A heurística ASP

Os bons resultados obtidos pela heurística *SP*, exposta na Seção 6.2.3, para instâncias onde existe abundância de banda nos servidores encoraja o uso de estratégias baseadas em

caminhos mínimos porém, a perda de desempenho desta mesma heurística em instâncias onde a banda não é abundante indica que ela deve ser melhorada.

Observando o grafo resultante da modelagem apresentada na Seção 6.2.4 exposta na Figura 6.10, é possível perceber que os únicos arcos que possuem custo não nulo são os arcos das classes b, c e d, no entanto, os arcos da classe c podem ser descartados, uma vez que os arcos desta classe possuem custos muito maiores que os da classe b e d, e portanto, nunca serão usados em uma solução do PFCM, uma vez que, qualquer quantidade de fluxo passando por arcos desta classe pode ser acomodada nos arcos da classe d sem prejuízo da função objetivo do problema.

Uma vez descartados os arcos da classe c e sabendo que os arcos das classes a, f e e possuem custo igual a zero, tudo o que é preciso fazer é usar os arcos da classe b de maneira adequada e, se necessário, escoar o restante das demandas pelos arcos da classe d. Com base nestes fatos é proposta uma nova heurística, exposta pelo Algoritmo 5, chamada de *Alternative Shortest Path* (*ASP*).

Algoritmo 5 Heurística *ASP*

```
1: Tratar requisições que solicitam conteúdos que possuem uma única réplica
2: Tratar requisições cujo o servidor origem possui o conteúdo solicitado
3: Construir uma lista com todas as requisições ainda não atendidas
4: Ordenar a lista em ordem decrescente de p_i
5: enquanto (Existir algum elemento na lista) faça
6:    Escolher uma requisição i na cabeça da lista
7:    Construir uma lista de servidores para atender i
8:    Ordenar a lista de servidores em ordem crescente de custo de atendimento
9:    Dividir&Atualizar(i , lista de servidores)
10:   se (A demanda de i não foi completamente atendida) então
11:      Enviar demanda restante para o servidor de backlog
12:   fim se
13:   remover a requisição i da lista de requisições
14: fim enquanto
```

A heurística *ASP* procede da seguinte maneira: primeiramente as requisições que solicitam conteúdos que possuem apenas uma réplica são tratadas. Este mecanismo previne que requisições que possam ser atendidas por outros servidores congestionem os servidores que abrigam as réplicas únicas antes que as requisições para estas réplicas possam ser tratadas. Após o tratamento das requisições por réplicas únicas, é feita uma tentativa para atender as requisições localmente, assim como na heurística *SP*, mostrada no Algoritmo 4. Em seguida, todas as requisições restantes devem ser inseridas em uma lista e esta deve ser ordenada em ordem decrescente de penalidade por *backlog*. Esta ordenação prioriza o

atendimento das requisições com maior penalidade na função objetivo, tentando reduzir estas penalidades. Em seguida, enquanto houver requisições na lista, devemos executar os seguintes passos: 1) Escolher a requisição i, de maior prioridade 2) Criar uma lista de servidores capazes atender a requisição i e ordenar esta lista por custo de atendimento. 3) Dividir a demanda da requisição i entre os servidores na lista de modo que os servidores de menor custo tenham prioridade no momento da divisão. Este passo é feito na linha 9 do algoritmo pela função *dividir&Atualizar(i , lista de servidores)* que, além de dividir a demanda entre os servidores de modo a reduzir o custo de atendimento, também atualiza os valores de banda utilizada por cada servidor. 4) Enviar, se necessário, demanda para o servidor de *backlog*. 5) remover a requisição i da lista de requisições.

Esta heurística, assim como a heurística *SP*, é baseada na estratégia de caminhos mínimos, no entanto, o grafo utilizado para calcular os caminhos mínimos não é o grafo de distâncias (utilizado pela heurística *SP*), e sim o grafo resultante da modelagem por fluxo em rede para o PDR. A prioridade dadas às requisições também é diferente visto que a ordem de chegada das requisições é desprezada pela heurística *ASP*, que prioriza as requisições de maior penalidade na função objetivo. Resultados experimentais mostram que a heurística *ASP* produz resultados de melhor qualidade do que *SP* contudo, a heurística *SP* apresenta resultados melhores em termos de tempo computacional.

6.3 Técnicas para Estimar a Demanda Futura

Estimação ou previsão é um tópico da estatística que tenta extrapolar observações feitas afim de determinar valores futuros para séries e variáveis. Várias técnicas são usadas para tais fins como por exemplo Redes Neurais Artificiais [32, 51], Algoritmos Genéticos [11] e modelos de Séries Temporais [34].

Alguns destes modelos são focados na precisão das estimativas, não se importando portanto com os tempos computacionais. Contudo, dentro de um ambiente com rígidas restrições de tempo, como um ambiente de rede, o tempo computacional gasto por um estimador de demanda também é um fator crítico, juntamente com a precisão. Assim, para que um estimador possa ser usado dentro do framework proposto nesta tese, ele deve despender pouco tempo computacional ao mesmo tempo que produz boas previsões. Assim sendo, apenas estimadores de baixos requerimentos em termos de tempo computacional foram utilizados nesta tese.

É importante mencionar que não existe modo de prever com exatidão a demanda

futura e nem um estimador que funcione melhor de maneira universal. Sempre há casos em que um estimador vai obter melhor desempenho que outros e casos onde este mesmo estimador pode não apresentar bons resultados. Assim, a busca de novos estimadores, mais precisos e rápidos, para o PPRDR tratado nesta tese é um ponto em aberto e será alvo de investigações futuras.

6.3.1 Conhecimento Futuro

O *Conhecimento Futuro* não é uma técnica de estimação, mas pode ser usado para avaliar a qualidade dos estimadores. Para tanto, basta ao invés da demanda estimada, usar a demanda real do próximo período.

Está claro que qualquer algoritmo que use este tipo de "estimador" não pode ser usado na prática, visto que é impossível saber a demanda futura com exatidão. No entanto, o conhecimento futuro pode ser usado para fornecer limites para as demais abordagens. Este conhecimento pode ser usado também para medir a precisão dos estimadores e o comportamento dos demais algoritmos (para resolução do PDR e do PPR) nos casos onde as demandas futuras são conhecidas, que se aplica a versão *offline* do PPRDR.

6.3.2 Estimador Baseado em Médias

O estimador baseado em médias, apresentado para o PPRDR em [38], foi uma tentativa simples para estimar a demanda futura. Este estimador prevê a demanda futura baseado na média aritmética das demandas passadas, sendo extremamente rápido, e de fácil implementação. Este tipo de estimador funciona bem em casos onde as demandas oscilam suavemente em torno de um valor constante, sendo um bom ponto de partida e também um bom parâmetro de comparação.

No entanto em casos onde a demanda oscilam abruptamente, este estimador não é adequado pois ele amortiza os valores das variações abruptas na demanda. Em [35] também foi constatado que este estimador sofre influência do número de períodos analisados. Por se tratar de uma média aritmética, demandas que existiram em períodos remotos influenciam no cálculo. Isto pode ser ruim em alguns casos, visto que o perfil das demandas pode mudar completamente de um período para outro dentro de uma RDC.

6.3.3 Estimadores Baseado em Alisamento Exponencial

De acordo com [34] a maioria dos métodos de previsão se baseia na idéia de que observações passadas contém informações sobre o padrão de comportamento da série temporal, e que o propósito dos métodos de previsão é distinguir este padrão de qualquer ruído presente nas observações, e utilizar este padrão para prever os valores futuros da série.

Ainda segundo [34] uma das grandes classes de métodos para previsão é a classe dos métodos de alisamento. Os métodos desta classe tratam os valores extremos das séries como aleatoriedade e, assim, através do alisamento destes valores extremos, é possível extrair o padrão básico da série.

Estes métodos são muito populares para previsões [8, 34] devido à relação entre simplicidade e precisão apresentada por eles. Além disso, como os cálculos executados por esta classe de métodos são muito simples, as previsões são feitas rapidamente, o que torna estes métodos bastante adequados para cenários onde o tempo é um fator crítico.

O método de Alisamento Exponencial, ou Alisamento Exponencial Simples (AES), utiliza uma função exponencial para alisar os valores extremos das séries, e pode ser descrito matematicamente por

$$z_t^{'} = \alpha z_t + (1-\alpha) z_{t-1}^{'}, z_0^{'} = z_1, t = 1, ..., |T| \tag{6.31}$$

ou

$$z_t^{'} = \alpha \sum_{k=0}^{t-1} (1-\alpha)^k z_{t-k} + (1-\alpha)^t z_0^{'}, t = 1, ..., |T| \tag{6.32}$$

onde $z_t^{'}$ é denominado valor exponencialmente alisado e α é chamada constante de alisamento, $0 \leq \alpha \leq 1$.

Como descrito, o AES pode ser visto como uma média ponderada que atribui pesos maiores às observações mais recentes, diferentemente, por exemplo, do método das médias móveis [34].

No entanto, o método de AES, quando aplicado a uma série que apresente tendências, fornece previsões que não refletem os valores reais. Para evitar este tipo de erro é necessá-

rio que o método seja adaptado para que possa responder mais adequadamente à presença destas tendências. Estas adaptações são conhecidas como Alisamento Exponencial Duplo (AED), nos quais os valores das tendências também são alisados exponencialmente. Dois métodos de AED serão apresentados nesta seção: O Alisamento Exponencial Linear de Brown (*AELB*) e o Alisamento Exponencial Biparamétrico de Holt (*AEBH*).

No *AELB*, um segundo valor exponencialmente alisado é utilizado, e pode ser calculado a partir da seguinte equação:

$$z_t^{''} = \alpha z_t^{'} + (1-\alpha) z_{t-1}^{''}, z_0^{''} = z_1, t = 1, ..., |T| \tag{6.33}$$

onde $z_t^{'}$ é dado pela equação (6.31).

A equação de previsão deste método pode ser dada por:

$$z_{t+r}^{p} = A_t + rB_t, \tag{6.34}$$

$$a_t = 2z_t^{'} - z_t^{''}, \tag{6.35}$$

$$b_t = \alpha/(1-\alpha)(z_t^{'} - z_t^{''}). \tag{6.36}$$

onde a_t e b_t são estimativas assintóticas de A_t e B_t, respectivamente, e r é o número de passos à frente de t que se deseja prever. Assim, a previsão pode ser dada por

$$z_{t+r}^{p} = (2 + r\alpha/(1-\alpha))z_t^{'} - (1 - r\alpha/(1-\alpha))z_t^{''}. \tag{6.37}$$

Para que estas equações possam ser utilizadas, deve-se estipular os valores inicias de $z^{'}$ e $z_t^{''}$. Em [8] o *AELB* foi utilizado para prever os valores de demandas para redes RDC, no entanto, os valores iniciais para estes termos não foram mencionados. Nesta tese os valores iniciais que foram utilizados são iguais à demanda real observada [34].

Um dos grandes problemas do *AELB* é a determinação do parâmetro α. Valores pequenos para α implicam em atribuir maior relevância às observações anteriores e vice-versa. Em [8] o *AELB* foi utilizado com parâmetro α fixo. No entanto, os valores fixos podem não ser a melhor opção dada a natureza dinâmica do problema. Os valores apresentados em [8] indicam que grande relevância deve ser dada às observações ocorridas no

passado. Entretanto, nos experimentos feitos nesta tese, foi constatado que a relevância dada aos valores apresentados nas observações passadas não deve ser maior que relevância dada às observações atuais. Um modo de resolver este impasse, e ao mesmo tempo tornar o método mais adaptativo às diversas situações que podem ser encontradas, é a técnica de *backforecasting*, exposta em [34], que consiste em utilizar o valores observados no período atual e no período passado para determinar o melhor valor possível para o parâmetro α. A técnica de *backforecasting* consiste em utilizar os valores do período anterior para prever os valores do período atual. Diversos valores para o parâmetro α são utilizados e aquele que apresentar o menor erro na previsão dos valores do período atual, que já são conhecidos, é selecionado para fazer a previsão dos valores para o próximo período. Uma vez que o parâmetro α é um valor real, apenas uma parte do conjunto de possíveis valores, que tem tamanho infinito, pode ser avaliada. Para esta tese, o conjunto de valores assumidos pelo parâmetro α estão dentro do intervalo $[0,1; 0,9]$ e variam com passo $0,1$.

Outro problema que pode ser destacado com o *AELB* é a restrição nas tendências lineares. Este método ajusta tanto o nível quanto a tendência com um mesmo parâmetro (α) de alisamento. Um método menos restrito que o AELB para lidar com séries que apresentem tendências é o Alisamento Exponencial Biparamétrico de Holt (*AEBH*) [34]. O *AEBH* é similar, em princípio, ao *AELB*. A principal diferença é que o *AEBH* alisa diretamente os valores da tendência. Isto permite grande flexibilidade uma vez que parâmetros diferentes de alisamento para o nível e para a tendência podem ser utilizados. As equações 6.38 e 6.39 dão os valores de nível e tendência no período t. Nestas equações α e λ são as constantes de alisamento e $z^{'}$ e D os valores exponencialmente alisados.

$$z_t^{'} = \alpha z_t + (1-\alpha)(z_{t-1}^{'} + D_{t-1}), \tag{6.38}$$

$$D_t = \lambda(z_t^{'} - z_{t-1}^{'}) + (1-\lambda)D_{t-1}, \tag{6.39}$$

A previsão para o períodos subsequentes é dada pela seguinte equação:

$$z_t^P = z_t^{'} + rD_t. \tag{6.40}$$

Como pode ser visto, a previsão é feita adicionando-se ao valor básico a tendência multiplicada pelo número de passos à frente que se deseja prever. Para utilizar estas equações valores iniciais devem ser usados para os fatores $z^{'}$ e D. De acordo com [34] um

modo simples e eficiente de determinar estes valores iniciais é atribuir a $D_1 = z_1 - z_0$ e $z_1^{'} = z_0$.

Para determinar as constantes de alisamento pode-se usar a técnica de *backforecasting*, assim como no *AELB*. A diferença é que para o *AEBH*, a técnica de backforecasting deve retornar um vetor (α, λ) que minimize o erro.

Este método, diferentemente do *AELB*, oferece uma maior liberdade por possibilitar que a tendência possa ser alisadas com uma constante diferente do nível, no entanto esta liberdade fornece mais um parâmetro a ser encontrado. O conjunto de valores testados para λ encontra-se dentro do intervalo $[0,1; 0,9]$ e tem incremento de $0,1$. O conjunto de valores usados para α foi o mesmo utilizado no método *AELB*.

Tanto o AELB e AEBH são utilizados para tentar prever a demanda dos clientes na resolução do PPRDR. As combinações que usam estes estimadores, bem como os resultados de computacionais apresentados por ambos serão discutidos mais adiante.

6.3.4 Estimador Baseado em Tendências

Este estimador é apresentado em [35] como uma alternativa simples, rápida e de fácil implementação e entendimento. Em [36], os autores buscavam entender a origem dos *gaps* obtidos pela heurística *HC* em relação à formulação *FD*. Uma das hipóteses, que veio a ser confirmada, e que a maior parte do *gap* é causada pelo processo de estimação da demanda futura. Os autores propuseram então um novo estimador chamado de *Based-on-Trends Estimator (*Bote*)* que procede de maneira extremamente simples. A demanda futura é prevista adicionando-se à demanda atual um fator Δ, que nada mais é do que a diferença entre a demanda atual e a demanda passada.

Este estimador é capaz de perceber mudanças nas tendências rapidamente uma vez que atribui a diferença das demandas, que é a amostra mais recente da tendência, ao valor da demanda corrente. Além disso, o uso da demanda atual como função base da previsão, em geral, tende a fornecer um valor mais preciso do que uma média por que as requisições que chegam no período atual provavelmente permanecerão no sistema da RDC por mais de um período.

Um dos pontos fortes deste estimador, em relação aos estimadores baseados em alisamento exponencial é que o estimador *Bote* não tem parâmetros a serem calibrados. Assim, para utilizar este estimador, nenhum processo auxiliar para calibrar parâmetros precisa ser feito. Além disso, este estimador pode ser usado desde o primeiro período,

ao contrário dos estimadores baseados em alisamento exponencial que necessitam de pelo menos duas medições para conseguir fazer uma previsão. Em relação ao estimador de médias, o *Bote* apresenta a vantagem de conseguir capturar rapidamente as mudanças na tendências, o que o estimador de médias simplesmente não consegue fazer em alguns casos.

6.4 Abordagens para o PPR

Nesta seção são apresentadas algumas contribuições desta tese e mecanismos propostos na literatura para a resolução do PPR que é um dos subproblemas do PPRDR.

6.4.1 Formulação Matemática para o PPR

Após analisar os resultados apresentados por heurísticas para o PPR [36], é constatado que existe a necessidade da construção de um método exato para confirmar algumas das hipóteses feitas em [36]. Ao se analisar o comportamento da heurística de replicação usada em [36], percebe-se que este algoritmo tenta inserir conteúdos em servidores onde estes conteúdos são mais requisitados, respeitando as restrições de espaço em disco. Em outras palavras, o algoritmo tenta inserir em um servidor, os conteúdos que trarão o maior benefício para aquele servidor respeitando a restrição de capacidade em disco. Uma mudança na terminologia do prolema contextualiza o PPR como uma variante do problema da mochila, também conhecido como *Knapsack Problem* [33]. Para tanto, basta mudar os termos servidor por mochila, espaço em disco por capacidade da mochila, demanda pelo conteúdo por benefício do objeto e tamanho do conteúdo para peso do objeto.

Dentro do contexto do problema da mochila, o problema de posicionamento de réplicas pode ser descrito como uma variante do Problema de Alocação Generalizado, que é descrito em detalhes em [33]. No problema de Alocação Generalizado, objetos com diferentes pesos devem ser colocados dentro de mochilas com capacidades heterogêneas. Cada par <mochila,objeto> possui um benefício associado de modo que o objetivo é decidir quais objetos levar em cada mochila de modo a maximizar o benefício total respeitando as capacidades das mochilas e o fato de que um objeto só pode estar em uma única mochila.

Uma formulação matemática para o Problema de Alocação Generalizado também pode ser encontrada em [33] e é aqui transcrita.

sejam:

- m o número de mochilas.
- n o número de objetos.
- p_{ij} o benefício de atribuir o objeto j à mochila i.
- w_{ij} o peso do objeto j quando associado à mochila i.
- c_i a capacidade da mochila i.

A formulação para o Problema de Alocação generalizado é dada por:

$$Max \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \tag{6.41}$$

$$S.a.$$

$$\sum_{j=1}^{n} w_{ij}x_{ij} \leq c_i, \quad i \in M = \{1, ..., m\}, \tag{6.42}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j \in N = \{1, ..., n\}, \tag{6.43}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in M, \forall j \in N. \tag{6.44}$$

A função objetivo, descrita em (6.41) maximiza o benefício dos objetos levados em cada uma das mochilas. As restrições (6.42) impedem que a capacidade da mochila seja excedida. As restrições (6.43) garantem que cada objeto esteja em exatamente uma mochila e as restrições (6.44) são as restrições de integralidade e não negatividade.

Esta formulação, com uma pequena modificação, pode ser utilizada para a resolução do PPR. Para isso, basta substituir o sinal $=$ nas restrições 6.43 por um sinal $\geq$. Assim sendo, o problema de Posicionamento de Réplicas pode ser resolvido pela seguinte formulação:

sejam:

- m o número de mochilas.
- n o número de objetos.
- p_{ij} o benefício de atribuir o objeto j à mochila i.
- w_{ij} o peso do objeto j quando associado à mochila i.
- c_i a capacidade da mochila i.

A formulação para o Problema de Alocação generalizado é dada por:

$$Max \quad \sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \tag{6.45}$$

$$S.a.$$

$$\sum_{j=1}^{n} w_{ij}x_{ij} \leq c_i, \quad i \in M = \{1, ..., m\}, \tag{6.46}$$

$$\sum_{i=1}^{m} x_{ij} \geq 1, \quad j \in N = \{1, ..., n\}, \tag{6.47}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in M, \forall j \in N. \tag{6.48}$$

A função objetivo, descrita em (6.45) maximiza o benefício dos objetos levados em cada uma das mochilas. As restrições (6.46) impedem que a capacidade da mochila seja excedida. As restrições (6.47) garantem que cada objeto esteja em pelo menos uma mochila e as restrições (6.48) são as restrições de integralidade e não negatividade.

Esta formulação, chamada de *AGAP*, pode ser utilizada para resolver o Problema de Posicionamento de Réplicas nos servidores, resultando na melhor alocação possível para uma demanda indicada, uma vez que se trata de um método exato. Isto ocorre porque a formulação *AGAP*, diferentemente da formulação para o Problema de Alocação Generalizado, permite que um objeto seja alocado em mais de uma mochila, restringindo unicamente que cada objeto esteja em pelo menos uma das mochilas. Isto se aplica ao PPR uma vez que um conteúdo pode ser replicado em mais de um servidor, sendo a única restrição que todo conteúdo possua pelo menos uma réplica, restrição esta que é atendida pela formulação *AGAP*. A formulação *AGAP* possibilita que o PPR, que até então vinha sendo resolvido heuristicamente [36, 38], seja agora resolvido de maneira exata.

Assim como a formulação *RF* a formulação *AGAP* encontra-se no capítulo de abordagens heurísticas pelo fato de ser usada para resolver um subproblema do PPRDR. Como a formulação AGAP é usada para resolver apenas parte do PPRDR, ela deve ser combinada com outros métodos, exatos ou heurísticos, para formar algoritmos híbridos, capazes de apresentar soluções para o problema. Logo, qualquer algoritmo que use a formulação AGAP para resolver o subproblema pode ser caracterizado como método híbrido.

6.4.2 Heurística de Descarte LRU

Esta heurística é utilizada por provedores reais de RDC [29] e é baseada no esquema *LRU* (*Least Recently Used*) [42] utilizado em sistemas operacionais para gerenciamento de páginas de memória. Uma vez que o espaço em memória principal se torna crítico, o sistema operacional escolhe algumas páginas desta memória para serem armazenadas na memória virtual do computador. Esta memória virtual nada mais é do que um espaço alocado em disco e que é tratado como depósito de páginas de memória. Um modo de determinar qual das várias páginas de memória armazenar na memória virtual é escolher a página usando a estratégia *LRU*. Neste modo, o sistema operacional elege para memória virtual páginas que foram usadas pela última vez em tempos mais distantes do tempo atual. Assim, o sistema operacional tenta manter em memória principal as páginas que estão possivelmente sendo usadas por programas em atividade.

A heurística de Descarte *LRU* utiliza um esquema similar ao utilizado nos sistemas operacionais como política de descarte de réplicas no caso de falta de espaço no disco dos servidores. Quando é constatada a necessidade de aquisição de um novo conteúdo por um servidor, o espaço livre no disco deste servidor é analisado. Caso o espaço livre em disco seja suficiente para acomodar o novo conteúdo, este novo conteúdo é replicado para o referido servidor sem necessidade de ajustes. Porém, no caso em que o espaço livre é insuficiente, conteúdos presentes no servidor precisam ser descartados em favor do novo conteúdo. A heurística de Descarte *LRU* escolhe para descartar os conteúdos que foram utilizados pela última vez em períodos mais distantes do período atual.

6.4.3 Heurística de Alocação por Servidor

Uma segunda abordagem não exata para o PPR é chamada de de Alocação Gulosa por Servidor (*Greedy Allocation per Server - GAS*) é utilizada em [35, 36, 38]. Esta heurística tenta inserir os conteúdos mais requisitados em cada servidor considerando a demanda local. Caso haja espaço suficiente em disco, os conteúdos são simplesmente replicados. Caso contrário, a heurística tenta remover réplicas dos conteúdos menos solicitados do servidor. O algoritmo *GAS* apresenta resultados competitivos com a formulação *AGAP* [35] além de apresentar uma implementação relativamente simples. Uma descrição mais detalhada desta heurística, juntamente com um pseudo-código de seu algoritmo serão apresentados a seguir.

Para resolver PPR, a heurística *GAS*, mostrado no Algoritmo 6 cria tuplas do tipo

$< cont, serv, custo >$, onde *serv* é um servidor da RDC, *cont* é um conteúdo não é armazenado em *serv* e *custo* é o custo de comunicação que *serv* paga por não ter uma réplica de *cont*, ou seja, é o custo de encaminhamento dos pedidos que solicitam *cont* para outros servidores da RDC. Estas tuplas são então armazenados em uma lista, chamada *insertCandidates*, e classificados em ordem decrescente de custo. O algoritmo itera sobre esta lista inserindo em cada passo, se possível, o conteúdo *cont* em *serv*. No caso em que não é possível inserir o conteúdo de *cont* em *serv* sem aumentar os custos de comunicação, a tupla é descartada. O algoritmo termina quando não houver nenhuma tupla restante na lista.

Como mencionado, a inserção de conteúdo pode ocorrer em dois casos: i) quando o servidor tem espaço suficiente para acomodar o conteúdo. ii) quando o servidor não tem espaço e um ou mais conteúdos são removidos, a fim de colocar o novo conteúdo. Este problema é tratado na linha 8 do algoritmo 6. Um conjunto de conteúdos, chamado *ContRem*, cujo a soma dos tamanhos é maior ou igual ao tamanho do novo conteúdo, é removido se o custo total de comunicação do conjunto *ContRem* for inferior ao custo de comunicação do novo conteúdo. Para construir este conjunto é criada uma segunda lista de tuplas, chamada *removeCandidates*, que contém todos os conteúdos já posicionados no servidor ordenados pelos custos de comunicação. O conjunto *ContRem* é inicializada como conjunto vazio. Os conteúdos são inseridos em *ContRem* seguindo a prioridade dada por *removeCandidates* até que a soma dos tamanhos dos conteúdos em *ContRem* seja maior ou igual ao tamanho do novo conteúdo.

Este procedimento de remoção pode levar a soluções inviáveis, pois em alguns casos ele remove todas as réplicas de um conteúdo do sistema da RDC. Isso ocorre quando um conteúdo é muito menos popular que os outros em todos os servidores. Nesse caso, é preciso reinserir o conteúdo removido em algum servidor da RDC, a fim de tornar a solução viável novamente. Nestes casos, a heurística *GAS* replica o conteúdo removido em seu servidor de origem.

6.5 Combinações

Esta seção apresenta algumas das várias combinações possíveis das heurísticas apresentadas que podem ser vistas na literatura [8, 29, 36, 38] e algumas que são propostas neste trabalho.

Algoritmo 6 Heurística GAS()

```
1: criar tuplas < cont, serv, custo > e armazenar em insertCandidates
2: ordenar insertCandidates em ordem decrescente de custo
3: para todo tupla < cont, serv, custo > em insertCandidates faça
4:    se existe espaço em serv para inserir cont então
5:       determinar o servidor mais próximo s de serv que tem uma replica de cont
6:       copiar cont em serv a partir de s
7:    senão
8:       tentar remover conteúdos de serv
9:       se remoção for bem sucedida então
10:         determinar o servidor mais próximo s de serv que tem uma replica de cont
11:         copiar cont em serv a partir de s
12:      fim se
13:   fim se
14: fim para
```

6.5.1 A Heurística *GHS*

Esta seção descreve uma solução de mercado usada em RDC reais, que é usada neste trabalho para comparação de resultados com as heurísticas propostas.

A heurística *Global Hosting System* (*GHS*) é uma solução patenteada de um provedor de RDC que é apresentada em [29]. Nesta heurística, as requisições não podem ser encaminhadas a outros servidores, sendo portanto, atendidas unicamente pelos respectivos servidores de origem. No caso em que o servidor de origem possui uma réplica do conteúdo desejado, o conteúdo é simplesmente transmitido para o cliente. Já no caso em que o servidor de origem não possui uma réplica do conteúdo desejado, um processo de replicação deste conteúdo é iniciado, sendo o atendimento da requisição postergado até que este processo de replicação esteja completo.

Para gerenciar os conteúdos nos servidores, um esquema *LRU* [42] é utilizado como política de descarte de réplicas no caso de falta de espaço no disco dos servidores para acomodar novos conteúdos. Caso o espaço em disco seja suficiente para acomodar os novos conteúdos, nenhum conteúdo presente no servidor é descartado.

Esta política, como descrito em [29], faz uso de pouco ou nenhum conhecimento sobre as demandas dos conteúdos, fazendo com que um mesmo conteúdo possa ser removido e replicado novamente diversas vezes ao longo do tempo. Como pode ser visto em [36] esta heurística não foi capaz de atingir resultados competitivos quando comparada com outras heurísticas. Isto ocorre porque esta heurística não foi proposta para lidar com restrições de QoS, fazendo portanto, um grande número de *backlogs*. Como os *backlogs* são duramente

penalizados na função objetivo considerada esta heurística acaba por produzir resultados de baixa qualidade.

No *framework* descrito neste capítulo, a heurística GHS pode ser descrita pelo Algoritmo 7.

Algoritmo 7 Heurística GHS()

1: **para todo** período de tempo **faça**
2: Algoritmo para o PDR: Atendimento Local
3: Estimador: Nenhum
4: Algoritmo para o PPR: Descarte LRU
5: **fim para**

Para poder fazer uma comparação justa entre as heurísticas propostas e a solução ofertada no mercado, um versão otimizada da heurística *GHS* é proposta. Esta nova heurística, chamada simplesmente de *OGHS* trata a replicação de conteúdos da mesma maneira que a versão original, no entanto as requisições são tratadas através do uso da formulação *RF*. Para isso, a heurística deve permitir que requisições sejam encaminhadas para outros servidores quando o servidor origem não possuir uma réplica do conteúdo desejado.

Como heurística *OGHS* usa uma formulação matemática para resolver parte do problema, ela também pode ser chamada de método híbrido, e como ela usa a formulação *RF* para resolver o PDR, comparar *OGHS* com as demais propostas não é mais uma comparação injusta visto que esta heurística também considera restrições de QoS na distribuição de requisições e difere das demais apenas no modo de tratar o PPR.

No *framework* descrito neste capítulo, a heurística OGHS pode ser descrita pelo Algoritmo 8.

Algoritmo 8 Heurística OGHS()

1: **para todo** período de tempo **faça**
2: Algoritmo para o PDR: Formulação *RF*
3: Estimador: Nenhum
4: Algoritmo para o PPR: Descarte LRU
5: **fim para**

Apesar de ser o primeiro algoritmo híbrido apresentado nesta tese, a heurística *OGHS* não é o primeiro algoritmo híbrido proposto para o PPRDR, sendo apresentada neste ponto apenas por questões didáticas. A primeira heurística híbrida proposta para resolver a versão *online* do PPRDR é a heurística *HC*, para qual a formulação *RF* foi desenvolvida

e, posteriormente, utilizada na construção de uma versão otimizada da solução utilizada em RDCs reais.

6.5.2 Heurística Construtiva Híbrida

A heurística *Hybrid Constructive* (*HC*), foi a primeira abordagem proposta para a versão *online* do PPRDR [38], que até então só havia sido abordado de maneira *offline* através da formulação matemática *FD*. A heurística *HC* é uma derivação da heurística *CORA* [46] introduzida para um problema semelhante PPRDR. A heurística CORA posiciona gulosamente réplicas em servidores baseando-se no custo de comunicação dos servidores, ou seja, número de requisições feitas a eles. Tendo o posicionamento das réplicas ela então resolve heuristicamente a associação das requisições aos servidores.

A heurística *HC*, assim como a heurística *CORA*, utiliza um algoritmo guloso para determinar o posicionamento das réplicas. Duas são as principais diferenças entre *CORA* e *HC*: A primeira é o uso de estimadores de demanda e a segunda o uso de um método exato para resolver a distribuição de requisições. Inicialmente, apenas o estimador de médias foi utilizado [38] na resolução do PPRDR. Observando que alguns trabalhos utilizam outros estimadores para problemas similares ao PPRDR [8], o uso de outros estimadores foi cogitado. Além do estimador de médias, usado em [36, 38], e do *AELB*, usado em [8], foram também utilizados o estimador *Bote*, *Conhecimento Futuro* e o *AEBH*. Também é preciso observar que apesar de ambas as heurísticas (*CORA* e *HC*), utilizarem algoritmos gulosos para resolver o PPR, os algoritmos utilizados são diferentes. O algoritmo utilizado pela heurística *CORA* supõe que existem réplicas fixas para cada um dos conteúdos, fato este que não ocorre no modelo tratado nesta tese. Por isso, o algoritmo *GAS* é utilizado como heurística para a solução do PPR.

O uso da formulação *RF* para a resolução do PDR é natural uma vez os estimadores de demanda, combinados com o algoritmo *GAS*, são capazes de fornecer um posicionamento de réplicas, que é a entrada de dados necessária para a execução da formulação *RF*. A importância desta heurística está no uso da formulação *RF*, que não só melhora o desempenho do algoritmo em termos de qualidade de solução como também torna esta heurística o primeiro método híbrido proposto para o PPRDR, sendo portanto a base para o desenvolvimento de várias outras heurísticas expostas nesta tese.

Em termos do *framework* descrito neste capítulo, a heurística *HC* pode ser descrita pelo Algoritmo 9.

Algoritmo 9 Heurística HC()

1: **para todo** período de tempo **faça**
2: Algoritmo para o PDR: Formulação *RF*
3: Estimador: Múltiplos
4: Algoritmo para o PPR: *GAS*
5: **fim para**

6.5.3 A Heurística *HCAGAP*

A heurística *HCAGAP* é apresentada em [35] com o objetivo de responder se os *gaps* entre *HC* e *FD* são ou não causados em sua maior parte pelo uso de um algoritmo guloso (*GAS*) como parte de *HC*.

Para isso, a formulação *AGAP* foi desenvolvida e usada como substituto do algoritmo *GAS* dentro dos moldes da heurística *HC*, dando origem à heurística *HCAGAP*. Assim, esta heurística resolve duas formulações matemáticas em cada período, sendo uma de minimização (*RF*) e outra de maximização (*AGAP*). O fato de resolver cada subproblema com um método exato não classifica este algoritmo como método exato uma vez que ele lida com o PPRDR através de decomposição em subproblemas, usa estimadores de demanda e resolve os subproblemas para cada período de tempo separadamente. Logo, este método também é classificado como método híbrido, já que conjuga simultaneamente conceitos de métodos exatos (formulações) e conceitos de métodos heurísticos (decomposição, previsão e divisão em períodos).

Assim como para a heurística *HC*, vários foram os estimadores utilizados: *Médias*, *Bote*, *Conhecimento Futuro*, *AELB* e *AEBH*. Esta heurística pode ser descrita, em termos do framework apresentado, pelo Algoritmo 10.

Algoritmo 10 Heurística HCAGAP()

1: **para todo** período de tempo **faça**
2: Algoritmo para o PDR: Formulação *RF*
3: Estimador: Múltiplos
4: Algoritmo para o PPR: Formulação *AGAP*
5: **fim para**

A importância da heurística *HCAGAP* reside no fato de que o uso de uma formulação matemática para resolver o PPR, possibilita que, dada uma previsão de demanda, obtenha-se o melhor posicionamento possível, quando é considerada somente a demanda local de cada servidor, fato este que não é possível de se garantir com o uso de algoritmos gulosos como a heurística *GAS*. Os resultados obtidos por esta heurística mostram que a principal causa dos *gap* apresentados por *HC* em [36] não é o uso da heurística gulosa

GAS para a resolução do PPR. Este fato foi uma das causas do estudo e da utilização de outros estimadores para o problema.

6.5.4 A Heurística *HNH*

Esta seção descreve a heurística *Hybrid Network Heuristic* (*HNH*), proposta pelos autores deste trabalho e apresentado em [35], que nos testes realizados até o momento se mostra a melhor heurística para a versão *online* do PPRDR em termos de qualidade de solução e uma das melhores em termos de tempo computacional.

Esta heurística é composta pelas melhores soluções para cada um dos subproblemas do PPRDR. Para resolver o PPR *HNH* usa a formulação *AGAP*, exposta na Seção 6.4.1 e usada nas heurísticas *HCAGAP*, que é capaz de fornecer a solução ótima para o PPR para uma certa demanda. Para prever as demandas futuras, vários estimadores foram usados: Conhecimento Futuro, Médias, Bote, AELB e AEBH. A resolução do PDR é dada pela solução do PFCM, em redes construídas pelo modelo de fluxo exposto na Seção 6.2.4, usando o algoritmo *Network Simplex* [7] disponível no resolvedor CPLEX [5]. Assim como a formulação *RF*, o modelo de fluxo é capaz de fornecer a solução exata para o PDR, no entanto este modelo demanda bem menos tempo computacional do que a formulação *RF*, como pode ser visto no Capítulo 8.

No *framework* descrito neste capítulo, a heurística *HNH* pode ser descrita pelo Algoritmo 11.

Algoritmo 11 Heurística HNH()

1: **para todo** período de tempo **faça**
2: Resolução do PDR: Modelo de Fluxo em Rede / Network Simplex
3: Estimador: Múltiplos
4: Resolução do PPR: Formulação AGAP
5: **fim para**

Esta heurística fornece as mesmas soluções que a heurística *HCAGAP* em um tempo computacional menor uma vez que a resolução do modelo de fluxo em rede é resolvido em menos tempo que a formulação *RF*. Assim, fica evidente dizer que a heurística *HNH* supera a heurística *HCAGAP* uma vez que fornece resultados iguais em um tempo computacional menor, como pode ser observado nos resultados expostos no Capítulo 8.

Um ponto importante a ser destacado é que o uso do modelo de fluxo em rede juntamente com a heurística *GAS* pode fornecer uma outra heurística híbrida para o problema, que fornece os mesmos resultados obtidos pela heurística *HC* em um tempo menor. Como

a análise do ganho de tempo já pode ser feita através da comparação entre as heurísticas *HC* e *HCAGAP*, considera-se desnecessário construir outra heurística para fazer a mesma análise para a heurística *HNH*, e portanto esta combinação (modelo de fluxo em rede + GAS) não será abordada.

6.5.5 Combinações Baseadas em Caminho Mínimo

Na literatura o PDR é muitas vezes considerado de fácil resolução e tratado através de abordagens baseadas em caminhos mínimos [8, 13, 25], que consideram apenas a distância entre os servidores para fazer a distribuição de requisições. Este tipo de abordagem ignora informações importantes presentes nas requisições, que podem ser usadas para melhorar a qualidade do serviço percebida pelo usuário final.

Esta seção apresenta três heurísticas baseadas em caminho mínimo para o PPRDR. Todas utilizam vários estimadores para a previsão de demanda: *Conhecimento Futuro*, *Médias*, *Bote*, *AELB* e *AEBH*. A diferença entre elas está na combinação de estratégias usadas na resolução do PPR e do PDR. Esta diferença entre combinações se deve principalmente pelo objetivo a serem atingidos no momento da construção de cada uma destas heurísticas, objetivos estes que serão explicados mais adiante nesta seção.

A primeira heurística a ser apresentada é a combinação da heurística *SP* para a resolução do PDR e da formulação *AGAP* para a resolução do PPR. Esta combinação recebe o nome de *SPA* e pode ser descrita, de acordo com o *framework* exposto no Algoritmo 3, pelo Algoritmo 12.

Algoritmo 12 Heurística SPA()

```
1: para todo período de tempo faça
2:    Resolução do PDR: Heurística SP
3:    Estimador: Múltiplos
4:    Resolução do PPR: Formulação AGAP
5: fim para
```

O objetivo da construção da heurística *SPA* é adaptar uma heurística da literatura [8] para o PPRDR e usá-la como parâmetro de comparação. A heurística proposta em [8] é uma boa escolha como semente para a adaptação devido aos bons resultados reportados e pelas similaridades entre os problemas. Ambas as heurísticas, *SPA* e a exposta em[8], utilizam o mesmo mecanismo de caminho mínimo para a resolução do PDR e utilizam estimadores baseados em alisamento exponencial. A diferença entre estas duas heurísticas, além do maior número de estimadores e do mecanismo de *backforecasting* utilizados por

SPA, está na abordagem de resolução do PPR. Em [8] é utilizado um algoritmo guloso de posicionamento de réplicas que supõe a existência de réplicas fixas em alguns servidores, fato este que, como dito anteriormente, não é permitido no modelo tratado nesta tese. Para obter um parâmetro mais justo de comparação ao invés de reimplentar o algoritmo guloso proposto em [8] e utilizá-lo nos testes computacionais sabendo que isto pode gerar divergências significativas nos resultados devido à diferença entre os modelos, optou-se por utilizar a melhor abordagem de resolução do PPR proposta nesta tese ($AGAP$) para compor, juntamente com as demais abordagens utilizadas em [8], uma versão adaptada para o PPRDR apresentada pela literatura.

A segunda heurística a ser apresentada é a combinação da heurística *ASP* para a resolução do PDR e da formulação $AGAP$ para a resolução do PPR. Esta combinação é chamada de *ASPA* e pode ser descrita pelo Algoritmo 13.

Algoritmo 13 Heurística ASPA()

```
1: para todo período de tempo faça
2:    Resolução do PDR: Heurística ASP
3:    Estimador: Múltiplos
4:    Resolução do PPR: Formulação AGAP
5: fim para
```

O objetivo da construção da heurística *ASPA* é fornecer uma abordagem baseada em caminho mínimo e que utilizasse os conceitos descobertos durante o desenvolvimento do modelo de fluxo em rede exposto na Seção 6.2.4. Para tornar esta heurística mais semelhante à heurística *SPA*, A estratégia de resolução do PPR e os estimadores foram mantidos, permitindo assim que o impacto das heurísticas *SP* e *ASP* na qualidade das soluções seja analisado de maneira mais consistente.

A terceira heurística apresentada nesta seção, ao contrário de *SPA* e *ASPA*, não é um método híbrido uma vez que não utiliza conceitos de métodos exatos. Trata-se portanto de uma versão completamente heurística para o PPRDR e é feita através da combinação do algoritmo *ASP* para a resolução do PDR e do algoritmo *GAS* para a resolução do PPR. Esta combinação, chamada o nome de $ALGA$, utiliza múltiplos estimadores, assim como *ASPA* e *SPA*, e pode ser descrita pelo Algoritmo 14.

O objetivo da construção da heurística ALGA é apresentar uma versão completamente heurística (que não utilize conceitos de métodos exatos) para o PPRDR. Dentre as heurísticas apresentadas para a versão *online* do PPRDR Alga é a mais rápida pois combina os elementos mais rápidos propostos nesta tese. É importante mencionar que uma versão mais rápida que a heurística $ALGA$ pode ser obtida através da combinação da

Algoritmo 14 Heurística ALGA()

1: **para todo** período de tempo **faça**
2: Resolução do PDR: Heurística ASP
3: Estimador: Múltiplos
4: Resolução do PPR: Formulação GAS
5: **fim para**

heurística *SP* e da heurística *GAS*. Entretanto, devido ao baixo desempenho apresentado pela heurística *SPA* em algumas instâncias onde existe uma maior restrição na banda dos servidores, optou-se por descartar esta combinação.

6.6 Algoritmos ILS para a Versão *Offline* do PPRDR

A versão *offline* do PPRDR é ainda pouco explorada pela literatura. Deste modo, uma versão da meta-heurística *Iterated Local Search* (*ILS*) [22, 44] é proposta nesta seção com o objetivo de servir como ponto de partida para futuros trabalhos com esta versão.

A idéia por trás da heurística *ILS* é construir um caminho aleatório no espaço de ótimos locais, definido como resultado da aplicação de um algoritmo de busca local [15]. Quatro componentes são necessários para a produção de um algoritmo *ILS*: 1) Um procedimento para gerar uma solução inicial, 2) uma busca local, 3) uma estratégia de perturbação e 4) um critério de aceitação de novas soluções.

A eficiência dos quatro componentes é importante no desempenho final da *ILS*, porém, a eficiência da busca local é decisiva uma vez que ela influencia fortemente na qualidade das soluções e no tempo computacional. A estratégia de perturbações deve permitir que o algoritmo *ILS* consiga escapar de ótimos locais ainda distantes de um ótimo global e, ao mesmo tempo, evitar as desvantagens de uma reconstrução total da solução, que por sua vez pode acarretar em uma possível desconsideração de informações relevantes contidas na solução corrente. A combinação da estratégia de perturbação com o critério de aceitação influencia fortemente no tipo de caminho que será percorrido dentro do espaço de soluções. Isto ocorre porque estes dois componentes são responsáveis pela diversificação do algoritmo e também pelo comportamento geral do algoritmo em relação às soluções encontradas ao longo do caminho.

Para melhorar o desempenho de um algoritmo *ILS* é necessário encontrar as melhores combinações de estratégias para os quatro componentes. Contudo, devido ao grau de interação entre estes componentes, encontrar as melhores combinações de estratégias

pode tornar-se uma tarefa difícil e, em muitos casos, estas combinações só podem ser determinadas heuristicamente [40].

Algoritmo 15 ILS

```
1: Solução s_0 = gerarSoluçãoInicial()
2: solução s* = buscaLocal(s_0)
3: enquanto critério de parada não for atingido faça
4:    Solução s' = perturbação(s*, históricoDeBusca)
5:    Solução s*' = buscaLocal(s')
6:    s* = critérioDeAceitação(s*, s*', históricoDeBusca)
7: fim enquanto
8: retorne s*
```

O Algoritmo 15 mostra o pseudo-código para um *ILS* geral. Primeiramente, uma solução inicial é gerada aleatoriamente ou utilizando um algoritmo construtivo [22]. A solução inicial é então submetida a um processo de refinamento através do uso de um algoritmo de busca local. Em seguida são repetidos os seguintes passos até que um critério de parada seja atingido: i) perturba-se uma solução semente encontrada anteriormente gerando uma nova solução. ii) refina-se a solução perturbada através de um procedimento de busca local. iii) A solução gerada nesta iteração é submetida a um critério de aceitação que verifica se a solução gerada nesta iteração é adequada para ser usada como semente em outras iterações.

Apesar de o histórico de busca estar presente nas etapas de perturbação e aceitação no algoritmo, indicando que este histórico pode ser usado tanto para intensificação quanto para diversificação nestas etapas, muitas vezes este histórico não é considerado devido à grande dificuldade em decidir quais atributos do histórico de busca utilizar. No entanto, o uso adequado de memória adaptativa em heurísticas e meta-heurísticas tem se mostrado um caminho efetivo na resolução de problemas de otimização [23, 40, 43] e portanto ignorar este histórico de busca pode levar à construção de algoritmos pouco efetivos. Neste sentido, são propostas nesta tese duas versões da meta-heurística *ILS*: uma que ignora completamente o histórico de busca, que será apresentada nesta seção, e uma que utiliza o histórico de busca para escapar de ótimos locais, apresentada na Seção 6.6.4. A seguir, serão discutidos os componentes utilizados na construção do *ILS* sem o uso do histórico.

6.6.1 Buscas Locais para o PPRDR

Ao longo do desenvolvimento desta tese, alguns mecanismos de busca local foram testados para fazer parte do algoritmo *ILS*, contudo, muitos deles mostraram-se inadequados devido à alta demanda por memória e/ou tempo computacional. Explorar a vizinhança de uma solução para o PPRDR não é uma tarefa trivial uma vez que o problema possui uma dimensão temporal. Para fazer mudanças no posicionamento das réplicas em um período t do horizonte de planejamento, é preciso recalcular os custos de replicação dos períodos $t-1$, que podem aumentar ou diminuir de acordo com o novo posicionamento em t e também os custos de replicação do período t, que podem aumentar ou diminuir de acordo com o posicionamento no período $t+1$. Isto ocorre porque os custos de criação de réplicas usadas em um dado período t não são pagos neste período, mas sim no período anterior, neste caso $t-1$. Além disso, uma vez que o posicionamento de réplicas é alterado em um período t também é necessário recalcular a distribuição de requisições para todos os períodos maiores ou iguais a t, fato este que dificulta a exploração exaustiva de uma ou mais vizinhanças. Assim sendo, são mostradas nesta seção três propostas de buscas locais: uma que se mostrou eficiente, sendo escolhida para utilização no algoritmo *ILS* e outras duas que apesar de fornecer bons resultados em termos de qualidade de solução, foram descartadas pela grande demanda por memória e/ou tempo computacional.

Dentre os algoritmos testados como busca local para o *ILS* apenas a meta-heurística Record-to-Record (RTR) [44] se mostrou capaz de fazer melhorias na qualidade das soluções em um tempo computacional viável, fato este que torna este algoritmo a escolha mais adequada para a busca local do *ILS*. A meta-heurística *RTR* é uma versão determinística da metaheurística *Simulated Annealing* [26] e vem sendo utilizada com sucesso para outros problemas de otimização [30, 31]. O algoritmo percorre a vizinhança de uma solução semente aleatoriamente e aceita um vizinho pior como nova semente desde de que o valor da função objetivo deste vizinho seja menor (para problemas de minimização) que o *Recorde* acrescido de um *Desvio*. O *Recorde* representa o valor da função objetivo da melhor solução encontrada até o momento. O *Desvio* é um valor que representa a distância máxima tolerada entre a melhor solução encontrada e uma solução à ser usada como semente. Assim, se o valor de *Desvio* for igual a zero, somente soluções que sejam melhores que a solução atual serão aceitas como semente. É importante ressaltar que o valor do *Desvio* influi diretamente na diversificação do algoritmo. Quanto maior o valor do desvio, mais soluções serão utilizadas como semente e, consequentemente, maior será a região do espaço de soluções explorada. O algoritmo 16 apresenta um possível pseudo-código para

a meta-heurística *RTR*.

Algoritmo 16 RTR(Solução s, Desvio)

```
1:  Recorde = funçãoObjetivo(s)
2:  Solução melhorSolução = s
3:  enquanto Critério de parada não for atingido faça
4:     Solução s' = vizinhoAleatório(s)
5:     se funçãoObjetivo(s') < Recorde + Desvio × Recorde então
6:        s = s'
7:     fim se
8:     se funçãoObjetivo(s') < Recorde então
9:        melhorSolução = s'
10:       Recorde = funçãoObjetivo(s')
11:    fim se
12: fim enquanto
13: Retorne melhorSolução
```

Para gerar o vizinho aleatório (linha 4 do Algoritmo 16) quatro estruturas de vizinhanças foram utilizadas: *Inserção de conteúdo*, *Remoção de conteúdo*, *Realocação de conteúdo* e *Mudança por popularidade*. Na *Inserção de conteúdo*, são escolhidos aleatoriamente um período t, um servidor s e um conteúdo k que não esteja alocado em s no período t. O conteúdo k é inserido em s no período t e conteúdos são removidos aleatoriamente até que a solução seja viável (sem violações no espaço em disco) tomando cuidado para que o conteúdo k não seja removido. Após o processo de remoção, é feita uma tentativa gulosa de inserção de conteúdos no servidor s no período t. Esta inserção é feita baseada no fato de que após a remoção aleatória, o servidor s pode ter espaço suficiente para acomodar outros conteúdos, deste modo tenta-se inserir conteúdos em s, priorizando aqueles conteúdos com maior popularidade global (número de requisições), sem que novas remoções sejam permitidas. Na *Remoção de conteúdo* são escolhidos aleatoriamente um período t, um servidor s e um conteúdo k alocado no servidor s no período t. O conteúdo k é removido caso seja possível e conteúdos são inseridos aleatoriamente em s no período t até que não haja mais espaço para acomodar nenhum conteúdo. Na *Realocação de conteúdos*, são escolhidos aleatoriamente um período t e dois servidores s_{origem} e $s_{destino}$. Em seguida computa-se um conjunto C de conteúdos presentes em s_{origem} e ausentes em $s_{destino}$ no período t. Caso o conjunto C seja vazio os papeis de s_{origem} e $s_{destino}$ são trocados. Um conteúdo $k \in C$ é escolhido aleatoriamente e tenta-se inseri-lo em $s_{destino}$, removendo conteúdos de maneira aleatória caso seja necessário. Caso não seja possível inserir o conteúdo k em $s_{destino}$, k é removido de C e outro conteúdo é escolhido aleatoriamente. O algoritmo procede até que seja encontrado um conteúdo que possa ser inserido em $s_{destino}$ ou até que o conjunto C esteja vazio. Na *Mudança por popularidade* são escolhidos aleatoriamente

um período t e um servidor s. De todos os conteúdos presentes em s no período t aquele com menor popularidade global e que passível de remoção (não é a única réplica de um conteúdos) é removido. Determina-se então qual é o conteúdo com maior popularidade global não alocado em s no período t e em seguida é feita uma tentativa de inserção deste conteúdo em s. Caso o espaço liberado pela remoção do conteúdo menos popular seja insuficiente outros conteúdos são removidos aleatoriamente.

É importante mencionar que todas as estruturas de vizinhança apresentadas podem falhar. A estrutura *Inserção de conteúdo*, por exemplo, pode falhar quando o servidor escolhido tiver réplicas de todos os conteúdos disponíveis ou quando a soma dos tamanhos dos conteúdos que podem ser removidos do servidor (não são réplicas únicas) é menor que o tamanho do conteúdo inserido. Quando uma falha ocorre, a estrutura de vizinhança escolhida é executada novamente até que seja produzido uma solução viável ou até um número máximo de três tentativas. Caso o número máximo de tentativas seja atingido, o algoritmo *RTR* prossegue, contabilizando a iteração atual como uma iteração onde nenhuma melhora foi alcançada. Nesta tese todas as estruturas de vizinhança possuem a mesma chance de serem escolhidas no momento de gerar o vizinho aleatório dentro do *RTR* e um número máximo de iterações sem melhora é utilizado como critério de parada do algoritmo, sendo este número dado por $(|T| + |C| + |S|/10) \times numPerturbações/2$ para as instâncias de pequeno porte (até 50 servidores) e por $|T| + |C|$ para as instâncias de grande porte (100 ou mais servidores).

Uma segunda opção a ser usada como busca local é um algoritmo que insere em um servidor um conteúdo mais popular, em termos de demanda global, que encontra-se ausente neste servidor em um determinado período de tempo. No caso em que não há espaço o suficiente para a inserção do conteúdo, todas as possíveis remoções de no máximo um conteúdo são analisadas de modo a tornar a inserção do novo conteúdo possível. Ao terminar a acomodação do novo conteúdo, os custos de replicação dos períodos envolvidos são atualizados e a distribuição das requisições refeita para todos os períodos necessários. Todas as possíveis inserções de conteúdos ausentes são analisadas para todos os servidores e todos os períodos. Este algoritmo não possui uma alta demanda por memória, no entanto, sua execução demanda muito tempo computacional. Uma única execução deste algoritmo para uma instância pequena pode levar horas, o que impossibilita o uso deste algoritmo como busca local do algoritmo *ILS*, onde a busca local é invocada diversas vezes.

Uma terceira opção é utilizar a formulação *FD* como busca local. Para fazer isso, basta fixar o valor de algumas variáveis de acordo com uma solução dada como entrada e

deixar que a formulação otimize o restante das variáveis. Contudo, não é fácil determinar de maneira eficiente quantas/quais variáveis fixar e quantas/quais deixar em aberto. Também é preciso levar em consideração que a formulação *FD* tem dificuldade em encontrar soluções para instâncias com grande número de servidores devido à sua grande demanda por memória. Um modo de resolver este problema é dividir heuristicamente o conjunto de servidores em vários subconjuntos e também dividir o horizonte de planejamento em sub-horizontes. Deste modo, é possível dividir as variáveis de decisão em vários subconjuntos e utilizar formulação *FD* para otimizar um destes subconjuntos enquanto os valores das variáveis de decisão pertencentes aos demais subconjuntos são fixados de acordo com a solução dada como entrada. A Figura 6.21 mostra como é feita a divisão dos conjuntos de servidores e períodos de tempo em subconjuntos sendo v o número de servidores em cada subconjunto de servidores e e o número de períodos em cada sub-horizonte.

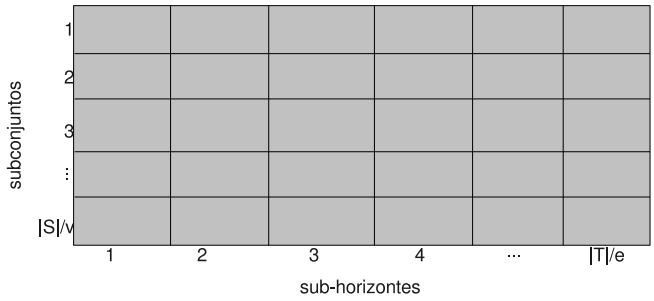

Figura 6.21: Divisão de Servidores e Tempo

É importante mencionar que os períodos dentro de cada sub-horizonte são consecutivos. Como exemplo seja $e = 5$. Assim o sub-horizonte 1 é composto dos períodos de 1 a 5, o sub-horizonte 2 é composto dos períodos de 6 a 10 e assim sucessivamente. Os subconjuntos de servidores já não possuem esta característica. Sendo $v = 3$ o subconjunto de servidores 1 pode ser composto por quaisquer três servidores, o subconjunto 2 pode ser composto de quaisquer três servidores exceto os presentes no subconjunto 1 e assim sucessivamente. Uma vez que todos os subconjuntos, de servidores e períodos, estão definidos pode-se percorrer a matriz de subconjuntos tanto por linha, otimizando assim um subconjunto de servidores para todos os períodos de tempo para só então otimizar os outros subconjuntos de servidores, ou por coluna, otimizando todos os subconjuntos de servidores para um mesmo sub-horizonte para só então passar para outro sub-horizonte.

Após definido o tipo de percurso a ser feito (por linha ou coluna) o algoritmo deterministicamente percorre a matriz de subconjuntos usando a formulação *FD* para resolver

variáveis relativas a um único quadrante e usando uma solução dada como entrada para fixar o valor das variáveis relativas aos outros quadrantes. O pseudocódigo deste algoritmo é mostrado no Algoritmo 17.

Algoritmo 17 FDBuscaLocal(Solução *sol*, e, v)

```
1: Dividir sub-horizontes de acordo com e
2: Dividir os servidores em subconjuntos aleatoriamente de acordo com v
3: enquanto Critério de parada não satisfeito faça
4:    i = 1,j = 1
5:    enquanto houver quadrante < i, j > não explorado faça
6:       Fixar as variáveis de FD para todos os quadrantes diferentes de < i, j > de
         acordo com sol
7:       Usar FD para resolver o quadrante < i, j > e atualiza sol
8:       Alterar os valores de i e/ou j de acordo com o percurso escolhido
9:    fim enquanto
10:   se não houve melhora em sol então
11:      Dividir os servidores em subconjuntos aleatoriamente de acordo com v
12:   fim se
13: fim enquanto
14: Retorne melhorSolução
```

O Algoritmo 17 divide o horizonte de planejamento em sub-horizontes de acordo com o parâmetro e. Em seguida monta os subconjuntos de servidores aleatoriamente de acordo com o parâmetro v. A seguir, repete os seguintes passos até que um critério de parada seja atendido: 1) Para todos os quadrantes $< i, j >$ fixa os valores das variáveis referentes aos demais quadrantes de acordo com a solução de entrada, resolve a formulação *FD* para otimizar $< i, j >$ e atualiza a solução de entrada caso necessário. 2) Após a otimização de todos os quadrantes, o algoritmo verifica se houve melhora na solução *sol*. No caso em que houve melhora, a otimização por quadrantes é executada novamente com os mesmos subconjuntos. Caso contrário, novos subconjuntos de servidores são gerados de acordo com o parâmetro v. O critério de parada utilizado nos testes computacionais feitos com o Algoritmo 17 é um número máximo de iterações sem melhora que foi configurado para $5 = e = v$.

Dentre os algoritmos de busca local analisados, *FDBuscaLocal* é o que apresenta as melhores soluções. Entretanto, este algoritmo apresenta alta demanda tanto em termos de memória quanto em termos de tempo computacional, o que também inviabiliza seu uso como busca local do *ILS*.

6.6.2 Critério de Aceitação, Solução Inicial e Critério de Parada

Devido aos bons resultados obtidos pela heurística *HNH*, exposta na Seção 6.5.4, tanto em termos de tempos computacionais quanto em termos da qualidade das soluções obtidas para o PPRDR, optou-se por utilizar esta heurística como método de geração inicial. Como na versão *offline* do PPRDR os dados de entrada são conhecidos *a priori*, não há incoerências no uso do estimador de *Conhecimento Futuro* neste caso e portanto este estimador pode ser utilizado na fase de previsão de demanda da heurística *HNH*.

Como critério de aceitação da meta-heurística *ILS* a estratégia elitista é utilizada. Esta estratégia que permite que apenas soluções melhores que a solução incumbente sejam aceitas para serem usadas como sementes em iterações posteriores. Como critério de parada é utilizado tempo total de processamento.

6.6.3 Estratégias de Perturbação

As perturbações possuem um papel crítico dentro do mecanismo de diversificação em um algoritmo *ILS*. Estas perturbações devem ser capazes de permitir que o algoritmo escape de ótimos locais evitando no entanto as desvantagens de um reconstrução total como por exemplo perda de informações relevantes que possam estar presentes na solução atual. A literatura recomenda que as perturbações utilizadas no *ILS* tenham um nível mais alto que as utilizadas nas buscas locais [22], ou seja, as perturbações do *ILS* devem provocar mudanças mais significativas na solução. Devido a isto, as perturbações usadas no algoritmo *RTR*, selecionado como busca local no *ILS* proposto, não são utilizadas como perturbações no *ILS*.

Ao todo, seis estratégias de perturbação foram propostas, contudo, somente três foram utilizadas. Segundo a literatura [22] uma boa perturbação deve transformar uma boa solução em um bom ponto de partida para uma busca local. Sendo assim, uma perturbação que não deve alterar apenas um pequeno número de componentes da solução de modo que o algoritmo de busca local possa desfazê-la facilmente e nem alterar um número excessivo de componentes de modo que solução produzida tenha pouca ou nenhuma semelhança com a solução original. O equilíbrio entre estes dois pontos é a chave para a construção de boas estratégias de perturbação. Neste sentido, três das perturbações propostas foram descartadas por fazer mudanças excessivas nas soluções. A seguir serão apresentadas as estratégias de perturbação utilizadas, sendo primeiramente apresentadas as estratégias que foram efetivamente utilizadas e em seguida as estratégias descartadas.

A primeira estratégia de perturbação descrita é uma combinação de todas as estruturas de vizinhança usadas dentro do algoritmo *RTR*. Nesta perturbação, um movimento de cada estrutura de vizinhança é feito na solução e só depois de todos os movimentos feitos, os cálculos referentes aos custos de replicação e distribuição de requisições são refeitos. As estruturas de vizinhança são aplicadas na seguinte ordem: *Adição de Conteúdo*, *Remoção de Conteúdo*, *Realocação de Conteúdo* e *Inserção por Popularidade*.

A segunda estratégia utilizada é a reconstrução aleatória do conjunto de réplicas de um servidor. Nesta perturbação, são selecionados aleatoriamente um período t e um servidor s. Todos os conteúdos de s são removidos e conteúdos são inseridos aleatoriamente até exaurir o espaço em disco do servidor. Esta perturbação pode gerar soluções inviáveis e, por tanto, após a reconstrução do servidor, um mecanismo de viabilização é utilizado antes de fazer a correção dos custos. O mecanismo de viabilização consiste em inserir conteúdos ausentes (sem nenhuma réplica) em servidores escolhidos aleatoriamente, removendo conteúdos aleatoriamente dos servidores escolhidos se necessário.

A terceira estratégia de perturbação utilizada é baseada em uma perturbação exposta em [22] que utiliza a meta-heurística *Simulated Annealing* (*SA*) para gerar novas soluções. Na perturbação exposta em [22] utiliza execuções curtas do *SA* para gerar soluções diferentes da solução de entrada. Esta perturbação, assim como outras estratégias de perturbação, pode gerar tanto soluções piores quanto soluções melhores que a solução de entrada. Entretanto, como o algoritmo por trás da perturbação é uma meta-heurística, as chances de gerar uma solução de melhor qualidade que a solução de entrada aumentam consideravelmente. Na perturbação proposta nesta tese, o algoritmo *RTR* é utilizado ao invés do *SA*, o que não afeta a natureza da perturbação visto que o *RTR* é uma versão determinística do *SA* [44]. Para tornar a execução do *RTR* curta, seguindo a política exposta em [22], o número de iterações sem melhora do *RTR* quando utilizado como perturbação é configurado para 10% do número utilizado como critério de parada do *RTR* quando usado como busca local.

A primeira estratégia descartada a ser discutida é similar à estratégia de reconstrução aleatória. Na perturbação descartada apenas um período é escolhido aleatoriamente e todos os servidores tem o conjunto de conteúdos reconstruídos aleatoriamente neste período. Os mecanismos de reconstrução e viabilização utilizados nesta perturbação são os mesmos descritos anteriormente.

Outra perturbação descartada é a troca entre períodos. Nesta perturbação, um período de tempo t é escolhido aleatoriamente. Também aleatoriamente é escolhido um

outro período que pode ser o período posterior ou anterior a t. O posicionamento de réplica destes dois períodos é trocado e as operações de viabilização e correção de custos realizadas depois da troca.

A última estratégia de perturbação descartada é chamada de cruzamento de períodos. Nesta perturbação, um período t é escolhido aleatoriamente e o posicionamento de todos os servidores neste período é descartado. Em seguida, os períodos $t+1$ e $t-1$ são usados para reconstruir o período t. Um dos períodos adjacentes à t é escolhido como base para os servidores de identificação impar e o outro período adjacente é escolhido como base para os servidores de identificação par. Suponha que o período escolhido como base dos servidores impares seja $t+1$. Assim, o conjunto de conteúdos de cada um dos servidores de identificação impar (1,3,5,...) no período t será uma cópias exatas do conjunto de conteúdos dos servidores de identificação impar no período $t+1$. Consequentemente, os servidores de identificador par no período t serão copias dos servidores de identificação par no período $t-1$. A escolha de qual período adjacente será usado como base para os servidores pares/impares é feita de maneira aleatória com a mesma chance para os dois períodos.

Uma última estratégia utilizada mas que não pode ser considerada uma perturbação é a ressubmissão da solução incumbente à busca local. Normalmente, estra estratégia não é considerada pelo fato de que a solução incumbente é um centro de atratividade e é resultado da submissão de outra solução à busca local. Isto quer dizer que, em muitos casos, a solução incumbente é um ótimo local e que simplesmente submeter esta solução à busca local novamente não traz benefícios. No entanto, como o *RTR*, usado como busca local nesta tese, não explora de maneira exaustiva a vizinhança de uma solução é possível que uma ressubmissão da solução incumbente possa resultar em uma solução de melhor qualidade, visto que o algoritmo pode na ressubmissão explorar uma parte da vizinhança que não havia sido explorada antes. Neste sentido, a ressubmissão da solução incumbente para a busca local pode também ser vista, de certo modo, como estratégia de intensificação, visto que este processo visa explorar de maneira mais apurada a vizinhança da solução incumbente. Assim sendo, a ressubmissão é utilizada no algoritmo *ILS* proposto nesta tese juntamente com as três primeiras estratégias de perturbação expostas nesta seção, sendo que à cada iteração do algoritmo *ILS*, uma destas estratégias (múltiplas perturbações *RTR*, reconstrução aleatória de um servidor, execução curta do *RTR* e ressubmissão da solução incumbente) é escolhida aleatoriamente sendo que todas as estratégias têm a mesma chance de serem escolhidas.

6.6.4 Um Algoritmo ILS com Memória Adaptativa para o PPRDR

Segundo a literatura [23, 43], o uso de memória adaptativa pode trazer melhorias na qualidade das soluções produzidas por algoritmos. Devido a este fato, é proposta nesta seção, um algoritmo *ILS* que faz uso de estruturas simples de memória adaptativa. O algoritmo faz uso de memória em dois níveis: dentro do *ILS*, através da alteração da probabilidade de escolha das estruturas de perturbação e alteração de parâmetros do *RTR*; E dentro do *RTR*, através de mudanças no critério de aceitação de novas sementes no algoritmo. A combinação destes dois mecanismos da origem a um novo algoritmo *ILS*, chamado a partir deste ponto de *ILSAM*. A seguir são apresentados a versão adaptativa do algoritmo *RTR* e, em seguida, o mecanismo de memória utilizado dentro do *ILSAM*.

6.6.4.1 Um Algoritmo RTR Adaptativo

O conceito de algoritmos adaptativos para diversos problemas de otimização não é recente [23, 43], entretanto, segundo o conhecimento dos autores, estes conceitos ainda são pouco aplicados para a meta-heurística *RTR*. Neste sentido é proposto nesta seção uma versão adaptativa da meta-heurística *RTR* que alterna entre fases de intensificação e diversificação.

O algoritmo proposto, chamado de *RTR-Adaptativo*, alterna entre três fases (Intensificação, Básico e Diversificação) através de mudanças no parâmetro *Desvio* do algoritmo *RTR* e pode ter seu comportamento descrito pelo diagrama da Figura 6.22.

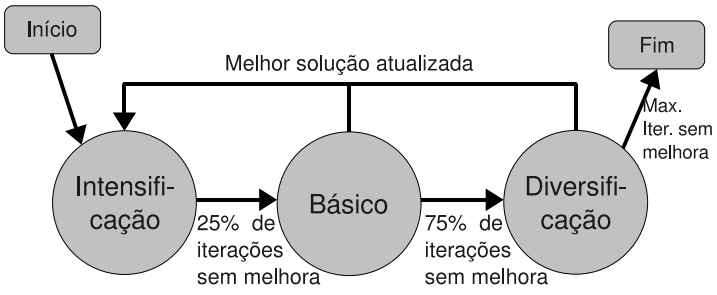

Figura 6.22: Modelo do algoritmo *RTR-Adaptativo*

Na fase de comportamento Básico o *RTR-Adaptativo* funciona exatamente como o *RTR* básico descrito na Seção 6.6.1. Na fase de Intensificação o *RTR-Adaptativo* tem o valor do parâmetro *Desvio* alterado para 0 (zero), o que faz com que apenas soluções melhores que a solução corrente sejam aceitas como novas sementes do algoritmo. Na fase

de Diversificação o parâmetro *desvio* tem seu valor aumentado em 10%, o que faz com que uma gama maior de soluções possa ser usada como novas sementes. O algoritmo inicia na fase de Intensificação devido ao fato de que, como o *RTR-Adaptativo* é usado como busca local do algoritmo *ILSAM*, é razoável pensar que as soluções passadas como entrada para o *RTR-Adaptativo* devem ter sua vizinhança melhor analisada. Caso sejam atingidas 25% do número máximo de iterações sem melhora o algoritmo passa para fase de comportamento Básico. Ao atingir 75% do número máximo de iterações sem melhora o algoritmo passa para a fase de Diversificação, onde uma vizinhança maior pode ser explorada. Sempre que uma nova melhor solução é encontrada o número de iterações sem melhora é zerado e o algoritmo volta à fase de Intensificação. As porcentagens de iterações sem melhora usadas como transição de fases do algoritmo foram determinadas experimentalmente e todos os demais parâmetros de execução tais como: o número máximo de iterações sem melhora, tipos de perturbação utilizadas, chance de escolha para cada perturbação e valor do *Desvio* são iguais para os dois algoritmos.

Testes computacionais, expostos no Capítulo 8 mostram que apesar da simplicidade, o algoritmo *RTR-Adaptativo* apresenta resultados muito bons uma vez que em 80% das instâncias testadas, o valor médio das soluções encontradas pelo *RTR-Adaptativo* é superior, em termos de qualidade, ao melhor resultado encontrado pelo *RTR* básico. Outro fato importante a ser mencionado é que o mecanismo adaptativo proposto para o *RTR* nesta tese não altera a complexidade das iterações do *RTR*. Segundo a literatura [23, 40, 43] em muitos casos o uso de memória adaptativa acaba por introduzir complexidade adicional às iterações dos algoritmos. Entretanto, as iterações dos algoritmos com memória adaptativa tendem a fornecer melhores resultados, o aumento na complexidade tende a ser compensado com um número menor de iterações. No mecanismo adaptativo proposto para o *RTR*, as iterações não sofrem alteração em sua complexidade visto que tanto no *RTR* normal como no *RTR-Adaptativo* os mesmos passos são seguidos. A diferença é o comportamento que a heurística tem em relação à aceitação de novas sementes. Na fase de Intensificação (até 25% de iterações sem melhora) o algoritmo tende a ser elitista, fazendo com que apenas a melhor solução encontrada até o momento seja usada como semente para geração do vizinho aleatório. Na fase de comportamento Básico (de 25% à 75% das iterações sem melhora), o algoritmo procede da mesma maneira que o *RTR* tradicional que permite que soluções piores que a melhor solução encontrada sejam utilizadas como semente para geração de novos vizinhos aleatórios porém, apenas soluções que estão até um certo limite da melhor solução são selecionadas como novas sementes. Na fase de Diversificação (de 75% à 100% das iterações sem melhora) o valor do limite

máximo aceito entre uma solução que é candidata à nova semente e a solução incumbente é aumentado, fazendo com que uma maior gama de soluções possa ser usada como semente e consequentemente aumentando a diversidade de soluções analisadas pelo algoritmo.

É importante relatar que apenas o algoritmo *RTR* usado como busca local utiliza o mecanismo de memória proposto, portanto, nenhuma alteração no comportamento do *RTR* é feita quando este é usado como estratégia de perturbação.

6.6.4.2 Uso de Memória no Algoritmo ILS

Nesta seção é descrito o mecanismo de memória usado dentro do algoritmo *ILSAM*, para alterar a probabilidade de escolha das estratégias de perturbação e para alterar parâmetros do *RTR*.

O mecanismos proposto, assim como o *RTR-Adaptativo*, é baseado em estados, porém ao invés de somente três estados como no *RTR-Adaptativo*, o mecanismo utilizado dentro do *ILSAM* possui quatro estados, sendo um de comportamento normal, um de intensificação e dois para a diversificação. A figura 6.23 mostra o diagrama de estados do *ILSAM*. O círculo com a letra I representa o estado de intensificação. O círculo com a letra N representa o estado normal e os dois círculos com a letra D representam os dois estados de diversificação. Note que toda vez que o algoritmo for alternar entre intensificação e diversificação é necessário passar pelo estado normal primeiro. A seguir serão descritos mais detalhadamente cada um dos estados bem como as transições entre eles.

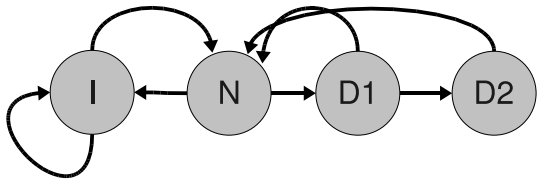

Figura 6.23: Diagramas de Estados do *ILSAM*

No estado normal, representado na Figura 6.23 por N, todos as estratégias de perturbação possuem a mesma chance de escolha. Este é o estado inicial do *ILSAM*. Estando o algoritmo ILS neste estado, há duas possibilidades de mudança uma para o estado I e uma para o Estado $D1$. Estando no Toda vez que o número de iterações executadas é

um múltiplo de 10, o algoritmo verifica qual foi a iteração em que a melhor solução foi encontrada. Caso a diferença entre a iteração em que a melhor solução foi encontrada e a iteração corrente seja maior que 10, o algoritmo muda para o estado $D1$. Caso contrário, ou seja, a diferença entre a iteração corrente e a iteração em que a melhor solução foi encontrada é menor que 10, as probabilidades de escolha das estratégias de perturbação são atualizadas e o algoritmo muda para o estado I.

No estado de intensificação, representado na Figura 6.23 por I, o mecanismo de escolha de perturbações utilizado é conhecido como mecanismo da roleta. Neste mecanismo cada um dos candidato recebe uma quantidade de fichas e estas fichas mudam de mãos de acordo com critérios estabelecidos. A chance de escolha de um candidato é proporcional à quantidade de fichas que ele possui, o que faz com que determinados candidatos tenham mais chance de ser escolhidos do que outros. Para esta tese, cada uma das estratégias de perturbação recebe inicialmente 25 fichas. Em todos os estados do algoritmo são computadas estatísticas que computam a quantidade total de tentativas, número total de tentativas antes da última transição de estado e número de vezes em que a melhor solução foi atualizada para cada uma das estratégias de perturbação. No estado de intensificação, um índice r que é dado pela divisão do número de atualizações da melhor solução pelo número de tentativas feitas desde a última transição de estado é calculado para cada estratégia de perturbação e utilizado para atualizar as chances de escolha de cada uma destas estratégias, de modo que as perturbações que possuem maior valor de r ganham fichas das perturbações que possuem menor r. Este cálculo de índices e troca de fichas é feito na transição entre os estados N e I e na transição de I para I, representada pelo auto-arco do estado I, de modo que a perturbação recebe duas fichas de cada perturbação que tenha o índice r menor que o seu. Com isso, as perturbações que tiveram melhor desempenho desde a última mudança terão mais fichas que as perturbações que não obtiveram um desempenho tão bom, o que aumenta a chance das perturbações de melhor desempenho serem escolhidas. Após a troca de fichas, todas as estatísticas referentes à última mudança de estados são zeradas. Isto é feito para que o algoritmo possa ter uma noção mais precisa do impacto das últimas alterações em seu comportamento. Assim, quando houver necessidade de refazer a análise para a próxima transição de estados, apenas dados referentes à probabilidades atuais serão considerados evitando que estatísticas calculas a muito tempo influenciem na decisão atual. É importante mencionar que, caso uma estratégia de perturbação não tenha sido usada desde a última mudança de estado, esta perturbação não sofre alteração no número de fichas e que não é permitido que nenhuma estratégia de perturbação tenha menos que 3 fichas. Estando no estado de

intensificação e sendo a iteração atual um múltiplo de 10 é feita a análise da transação de estados. Caso uma nova solução incumbente tenha sido encontrada nas últimas 10 iterações o algoritmo permanece no estado I, fazendo as atualizações necessárias nas chances de escolha das perturbações. Caso contrário, o algoritmo passa para o estado N, onde todas as perturbações possuem a mesma chance de escolha.

No estado $D1$ todas as perturbações possuem a mesma chance de serem escolhidas. Neste estado, o que muda é o valor do parâmetro *Desvio* do algoritmo *RTR*, sendo este alterado para 125% de seu valor original. Cabe lembrar que dentro do algoritmo *RTR*, na sua versão adaptativa, o valor do parâmetro *Desvio* também é alterado. Isto permite que o algoritmo $RTR - Adaptativo$ possa analisar vizinhança ainda maior, visto que, o valor de *Desvio*, aumentado em 25% pelo *ILS*, é aumentado novamente em 10% pelo *RTR-Adaptativo*. Estando neste estado para mudar o estado $D2$ são necessárias 10 iterações sem melhora. Caso uma solução melhor que a solução incumbente seja encontrada o algoritmo retorna ao estado N. O estado $D2$ é idêntico ao estado $D1$, porém ao invés de aumentar o valor de *Desvio* em 25%, o estado $D2$ aumenta este valor em 50%.

Como dito anteriormente, ao combinar o uso do mecanismo proposto para gerenciamento das probabilidades de escolha das perturbações e mudança do parâmetro *Desvio* do algoritmo *RTR*, exposto nesta seção, com o *RTR-Adaptativo*, exposto na Seção 6.6.4.1, obtém-se um novo algoritmo, chamado de *ILSAM*, que é mais eficiente quando comparado com a versão básica do *ILS*. Todos os demais componentes do *ILSAM*, tais como, critério de parada, critério de aceitação, entre outros, são os mesmos utilizados pelo *ILS* e os resultados obtidos por estes dois algoritmos estão expostos no Capítulo 8.

6.6.5 Ordem cronológica da Construção das Heurísticas

A ordem em que as heurísticas são apresentadas no texto, como dito anteriormente, não é a ordem cronológica de desenvolvimento das mesmas. As heurísticas estão agrupadas ao longo do texto de modo que as heurísticas com maior similaridade entre si fiquem próximas umas das outras. Contudo, também por razões didáticas, mostrar a ordem cronológica da construção destas heurísticas pode ser enriquecedor no processo de entendimento desta tese. Assim, esta seção apresenta os diversos métodos propostos na sua devida ordem cronológica.

Após algum tempo analisando o PPRDR na sua versão estática, estabeleceu-se que o problema atacado nesta tese seria a versão dinâmica e *online*. Entretanto, devido à necessidade de obter soluções ótimas para comparação, a primeira abordagem a ser

desenvolvida foi a formulação matemática *FD*.

Em seguida, com o objetivo de fornecer uma primeira abordagem para a versão *online* do problema é construída uma versão rudimentar da heurística *HC* apresentada nesta tese, que incluía apenas o estimador de médias e não conseguia fornecer soluções de boa qualidade para algumas instâncias. Ao longo do tempo, tanto o mecanismo de previsão de demandas através do uso de novos estimadores como a heurística *GAS*, foram sendo corrigidas e melhoradas até chegarem aos resultados aqui apresentados.

A terceira abordagem a ser construída é a heurística *GHS* utilizada por RDC reais e em seguida, a heurística *OGHS*.

A heurística *HCAGAP* foi construída logo após a publicação dos resultados obtidos por uma nova versão da heurística *HC* e da heurística OGHS [36] com o objetivo de tentar identificar a causa dos *gaps* de *HC*. A construção desta heurística motivou o uso da formulação *AGAP* em outras heurísticas bem como o uso de outros estimadores para o problema.

A heurística *SPA* é a sexta heurística a ser construída. O objetivo por trás da construção desta heurística é criar uma versão adaptada de um algoritmo apresentado pela literatura para que pudesse servir de base para comparação. Os bons resultados obtidos por esta heurística motivaram a criação de uma nova classe de instâncias bem como encorajaram a construção de novas modelagens baseadas em caminho mínimo.

Durante uma tentativa de melhoramento da heurística *SPA*, foi percebido o potencial do PDR para ser resolvido por algoritmos de fluxo em rede. Deu-se início então a construção e avaliação de uma série de modelos matemáticos que resultaram no modelo de fluxo em rede apresentado na Seção 6.2.4. A criação do modelo motivou a construção de uma heurística que o utilizasse, dando origem à heurística *HNH*.

Após a construção de *HNH* foi construída a heurística *ASP* para resolução do PDR e consequentemente as heurísticas *ASPA* e *ALGA*.

O algoritmo *SEA* foi feito após a construção das heurísticas para a versão *online* e as versões da meta-heurística *ILS* foram feitas por último.

Capítulo 7

Instâncias de Testes

Devido ao desconhecimento, por parte do autor, acerca da existência de instâncias que contemplem todas as características do problema aqui enfocado, foram geradas para testes instâncias com números variados de servidores, conteúdos e requisições.

O processo de geração das instâncias é dividido em partes. O primeiro passo é a geração da topologia dos servidores. Para esta parte do processo foi utilizado o gerador de topologias BRITE [4], amplamente difundido na comunidade científica e capaz de gerar topologias para diversos tipos de rede. Foram geradas ao todo quatro topologias utilizando o modelo *Waxman* para sistemas autônomos, com 10, 20, 30, 50, 100, 200, 300, 400 e 500 servidores. Para todos os parâmetros utilizados no processo de geração de topologia são utilizados os valores padrão, fornecidos pelo BRITE, exceto dois: os parâmetros N (número de nós na topologia) e o parâmetro M (número de conexões feitas por cada nó). O parâmetro N foi configurado para produzir topologias com 10, 20, 30, 50, 100, 200, 300, 400 , 500 e 600 servidores, como dito anteriormente. O parâmetro M foi configurado para 2 nas topologias de 10 servidores e para 3 nas demais topologias.

A Figura 7.1 ilustra uma tomada de tela do gerador de topologias BRITE mostrando os valores padrões utilizados no processo de geração de topologias. Em destaque, estão os campos utilizados para configurar os parâmetros N e M, os quais tiveram os valores padrões alterados para gerar as topologias deste trabalho.

Tendo as topologias, que informam quais os servidores se comunicam entre si, o próximo passo é gerar as demais informações do problema para criar as instâncias. Foram criados, até o momento, quatro classes de instâncias, chamadas de classes A, B, C e D. A seguir é mostrado como foram geradas as instâncias com até 50 servidores.

As instâncias da classe A são instâncias em escala reduzida, utilizadas para testes.

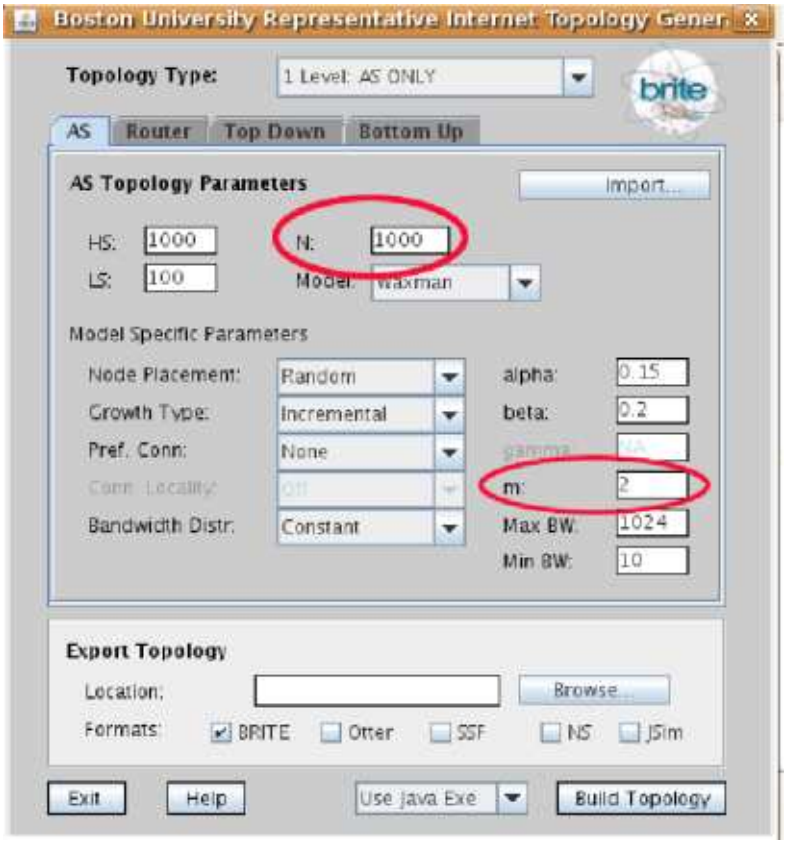

Figura 7.1: Tomada de Tela do gerador de topologias BRITE

As Instâncias da classe B são instâncias construídas com base em valores encontrados na literatura para problemas semelhantes ao abordado nesta tese. As instâncias da classe C são instâncias idênticas as instâncias da classe B, contudo, nas instâncias da classe C os servidores possuem restrições mais rígidas na capacidade de armazenamento. As instâncias da classe D são instâncias com restrições mais severas em termos de capacidade de armazenamento e em termos de banda nos servidores. Foram construídas 5 instâncias para cada par $<classe, número\ de\ servidores>$ até o tamanho de 50 servidores. Para instâncias com 100 ou mais servidores, foram escolhidas apenas as classes C e D.

Para gerar as demais informações das instâncias foi utilizado um gerador, construído em C++. Este gerador utiliza como parâmetro de entrada parte da topologia gerada pelo BRITE e utiliza algumas distribuições de probabilidade para gerar as demais informações.

A primeira informação a ser determinada é o número de períodos que serão considerados no processo de otimização. São considerados 15 períodos de tempos para as instâncias da classe A e 35 para as demais classes.

Em seguida são geradas as informações dos enlaces da rede. O arquivo contendo a

topologia é lido e um grafo de adjacência é construído a partir deste arquivo. Para cada aresta deste grafo é atribuído um atraso aleatório que varia entre 60 e 100 milissegundos. Em seguida é executado o algoritmo de Dijkstra encontrando as distâncias mínimas entre cada par de servidores. Estas distâncias são escritas em um arquivo como sendo as distâncias entre os servidores no primeiro período de tempo.

A cada cinco períodos de tempo, um dos enlaces tem seu atraso reconfigurado para um valor entre 60 e 100 ms. Como mudanças em um enlace podem acarretar mudanças em todas as distâncias mínimas no grafo, o algoritmo de Dijkstra é aplicado novamente e os valores de distância que tenham sido modificados são escritos em disco juntamente com o período em que ocorreu a modificação. Existem maneiras mais eficientes de se calcular as atualizações nos caminhos mínimos de um grafo [17] [47], entretanto, como o foco deste trabalho não são estas atualizações, por simplicidade, optou-se por reutilizar o algoritmo de Dijkstra.

Para as classes A, B e C, a rede possui canais simétricos, ou seja, o tempo que um pacote leva para ir de um servidor j para um servidor l é igual ao tempo gasto para ir de l para j. Para as instâncias da classe D a rede possui canais assimétricos, o que significa que o tempo gasto para um pacote ir de j para l é diferente do tempo gasto para ir de l para j. Esta assimetria nos canais torana as instâncias da classe D mais próximas dos casos reais, uma vez que estas assimetrias estão presentes em redes de computadores reais.

O próximo passo a ser dado é gerar as informações dos servidores. Primeiramente, a cada servidor é atribuído um identificador único. Em seguida, para gerar o espaço em disco em cada servidor foi utilizada a distribuição uniforme de probabilidade com os valores entre 100 e 200 MB para as instâncias da classe A, 100 e 150 GB [52] para a classe B, 3 e 4 GB para a classe C e $2,5$ e $3,2$ GB para a classe D. Para gerar a banda de cada servidor também 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.

Após a geração dos servidores, é feita a geração das informações dos conteúdos. Para gerar estas informações, a primeira tarefa a ser feita é 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 RDC antes do período final, e também atribuir a cada conteúdo um identificador único. Neste trabalho estabeleceu-se que o número de conteúdos permanentes a ser utilizado é de 3 conteúdos para as instâncias da

classe A, 10 para as instâncias das classes B e C e de 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. Outra informação relacionada aos conteúdos permanentes e voláteis são os períodos de surgimento e remoção destes conteúdos. Para os conteúdos permanentes, esta informação é determinada de maneira trivial, sendo o período inicial de simulação o período de surgimento do conteúdo e o período final sendo o período de remoção dos mesmos. Para os conteúdos voláteis, é estabelecido que um conteúdo pode surgir entre o período inicial e o quinto período subsequente a ele. Para o período de remoção é estabelecida uma relação semelhante, podendo um conteúdo ser removido em qualquer período entre o período final e o quinto período anterior ao período final. Para determinar o tamanho dos conteúdos são utilizados números aleatórios entre 10 e 20 MB para os conteúdos da classe A e entre 250 e 400 MB para as classes B e C. Para a classe D o tamanho dos conteúdos varia entre 350 e 450. Cada conteúdo possui um servidor origem, ou seja, um servidor no qual este conteúdo se originou. Neste parâmetro, a cada conteúdo é atribuído um servidor de maneira aleatória tendo todos os servidores a mesma chance de escolha.

Para gerar as requisições devemos ter em mãos as informações dos conteúdos devido ao fato de um número maior de requisições é dado aos conteúdos mais populares [14] [52] e também para não gerar requisições para conteúdos que não se encontram disponíveis na RDC. Por conteúdos não disponíveis entenda-se conteúdos que ainda não foram criados e/ou conteúdos que já foram removidos. Informações sobre o número de servidores também são necessárias já que, quanto maior o número de servidores, maior o número de pontos de entrada para a RDC 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.

Um fato conhecido é o de que alguns conteúdos são mais populares que outros, e que isto implica em um maior número de requisições para estes conteúdos mais populares. Tendo estes conceitos por princípio, estabeleceu-se que os conteúdos permanentes seriam mais populares que os conteúdos voláteis, e dentro de cada uma destas duas classes, os conteúdos com menor número de identificação seriam mais populares. Para gerar o número de requisições por período de tempo, foi decidido que este número seria arbitrário, podendo inclusive ser zero, para as instâncias da classe A. Para as classes B, C e D o número de requisições por período é descrito pela seguinte equação:

$numT = rand(20, 25) \times numeroServidores/5$, onde $numT$ é o número total de requisições em um período. Tendo o número de requisições para um período de tempo, deve-se determinar como as requisições serão divididas entre os conteúdos. Segundo [49] e [16] as requisições de busca para múltiplas chaves tendem a seguir um padrão de distribuição que se assemelha à distribuição Zipf [54]. Essa distribuição diz que os chaves mais populares tendem a concentrar a imensa maioria do número de requisições. Tendo este princípio em mente e sabendo quais os conteúdos mais populares foi decidido que, para as instâncias das classes A, B e C, cada conteúdo, exceto o menos popular receberia a uma fração das requisições de acordo com a seguinte equação: $nr_k = numT \times (NC - (p_k - 1))/\sum_{s=1}^{NC} s$. Onde NC é o número total de conteúdos e p_k é a ordem em que o conteúdo k está em relação a popularidade. Por exemplo, para o conteúdo mais popular este valor seria 1, para o segundo mais popular, seria 2 e assim sucessivamente. Como dito, a equação mostrada anteriormente é utilizada para determinar o número de requisições de todos os conteúdos exceto o menos popular. Uma vez determinado o número de requisições para os demais conteúdos, ao conteúdo menos popular caberão as requisições que faltam para completar o número total de requisições. Por exemplo, suponha que em um período existam 100 requisições e 3 conteúdos. Assim, seguindo as equações mostradas, o conteúdo mais popular ficaria com 50 requisições, o segundo mais popular ficaria com 33 requisições e o terceiro ficaria o restante das requisições, que seria, neste caso, 17 requisições. Para as instâncias da classe D, não há uma fração definida das requisições para cada conteúdo. As requisições são atribuídas aos conteúdos de acordo com uma distribuição Zipf como descrito em [16] com parâmetro $\alpha = 0,75$.

Para completar as informações das requisições são atribuídos a elas atrasos locais, que representam o tempo que os pacotes levam para transitar entre o cliente e o servidor ao qual o cliente está conectado. Estes atrasos variam entre 50 e 100 ms para as as classes A, B e C. Para a classe D os atrasos locais encontram-se entre 250 e 300. As informações referentes à qualidade de serviço exigida são o atraso máximo, a banda mínima e a banda máxima. O atraso máximo permitido varia entre 400 e 800 ms para todas as classes de instâncias, já a banda mínima assume os valores 5 ou 6 MB por período para as instâncias da classe A, e varia entre 40 e 42 MB por período para as classes B, C e D. A banda máxima varia entre 7 e 10 MB por período para a classe A e 45 e 50 MB por período para as instâncias das classes B, C e D. A cada requisição também é atribuído um servidor aleatoriamente de modo que todos os servidores possuem a mesma chance de escolha. Cada requisição também sabe o período em que ela será recebida pela RDC.

Após a realização de vários testes computacionais, foi constatado que as instâncias

das classes A e B são de fácil resolução, como pode ser visto no Capítulo 8. Sendo assim, para as topologias de grande porte (com mais de 100 servidores) apenas instâncias pertencentes à classes C. Cada possível tamanho de topologia possui 5 instâncias , como há 6 tamanhos possíveis (100, 200, 300, 400, 500 e 600) no total 30 instâncias de grande porte foram geradas. Nos testes computacionais realizados para esta tese só foi possível tratar instâncias com até 400 servidores devido à grande demanda por memória dos métodos exatos e híbridos e portanto resultados para as instâncias de 500 e 600 servidores não serão abordados.

Para melhor analisar os resultados das meta-heurísticas *ILS* e *ILSAM* também foram criadas instâncias da classe D com 100 servidores, no entanto, devido ao reduzido número de iterações das meta-heurísticas que é possível realizar no tempo limite estabelecido, julgou-se desnecessário criar instâncias da classe D com mais de 100 servidores visto que os resultados para instâncias maiores que estas obtidos pelas meta-heurísticas são pouco conclusivos, como é mostrado no Capítulo 8.

Capítulo 8

Resultados Computacionais

Este capítulo apresenta os resultados computacionais das abordagens exatas e heurísticas apresentadas nos Capítulos 5 e 6.

Os algoritmos apresentados nos capítulos anteriores estão implementados na linguagem C++ e compilados com o compilador g++ versão 4.3. As execuções são feitas em um computador Quad-Core com 2.83 GHz por core, 8 Gigabytes de RAM rodando um sistema operacional Linux com kernel 2.6. Para resolver as formulações matemáticas, tanto a *FD* quanto *RF* e demais é utilizado o resolvedor CPLEX 11.2 [5]. Para resolver os problemas de fluxo em rede construídos através do modelo apresentado na Seção 6.2.4 também foi utilizado o resolvedor CPLEX, que possui uma versão do *Network Simplex*.

Os testes foram divididos em duas baterias. Primeiro foram feitos testes com as instâncias de pequeno porte (até 50 servidores) e, após análise detalhada, as abordagens mais adequadas são utilizadas para as instâncias de grande porte.

8.1 Resultados das Abordagens Exatas

Esta seção apresenta os melhores resultados encontrados pelas abordagens exatas para o PPRDR, apresentadas no Capítulo 5. Estes resultados são usados como referência para análises de *gap* e tempos computacionais.

A Tabela 8.1 mostra os melhores resultados exatos para as instâncias de pequeno porte da classe A. Na primeira coluna estão os nomes das instâncias. A segunda coluna apresenta o tempo computacional, em segundos, necessário para encontrar a solução ótima. A terceira Coluna expõe o valor da função objetivo da solução encontrada. A quarta e última coluna apresenta o método que encontrou a solução. Note que para esta classe

de instâncias a formulação *FD* obteve os melhores resultados em todos os casos. Outro ponto a ser destacado é o fato de que é possível resolver instâncias de vários tamanhos em um tempo computacional relativamente curto.

Instância	Tempo (s)	F.O.	Método
10A01	1.21	300268	FD
10A02	1.04	272965	FD
10A03	1.15	253842	FD
10A04	1.28	349568	FD
10A05	0.84	265562	FD
20A01	3.47	2.00E+006	FD
20A02	8.93	2.14E+006	FD
20A03	7.5	2.07E+006	FD
20A04	6.76	2.08E+006	FD
20A05	6.23	2.10E+006	FD
30A01	20.77	2.10E+006	FD
30A02	14.86	2.16E+006	FD
30A03	15.25	2.15E+006	FD
30A04	17.51	2.22E+006	FD
30A05	19.12	2.24E+006	FD
50A01	59.98	463848	FD
50A02	63	821229	FD
50A03	62.16	878728	FD
50A04	60.19	649137	FD
50A05	58.57	754288	FD

Tabela 8.1: Melhores Resultados Exatos: Classe A

A Tabela 8.1 mostra os resultados obtidos para as instâncias de pequeno porte da classe B. As colunas desta tabela possuem o mesmo significado das colunas da tabela 8.1. Também para esta classe a formulação *FD* apresenta melhores resultados. É importante ressaltar que apesar de as duas classes (A e B) possuírem instâncias com o mesmo número de servidores, o CPLEX apresenta tempos computacionais maiores para resolver as instâncias da classe B. Isto ocorre principalmente devido ao maior número de requisições apresentado pelas instâncias desta classe.

Instância	Tempo (s)	F.O.	Método
10B06	6.02	2.00E+006	FD
10B07	5	2.14E+006	FD
10B08	4.86	2.07E+006	FD
10B09	4.98	2.08E+006	FD
10B10	5.07	2.10E+006	FD
20B06	29.43	4.20E+006	FD
20B07	27.58	4.33E+006	FD
20B08	31.04	4.29E+006	FD
20B09	29.43	4.43E+006	FD
20B10	28.57	4.48E+006	FD
30B06	92.25	6.56E+006	FD
30B07	80.79	6.34E+006	FD
30B08	73.5	6.56E+006	FD
30B09	72.12	6.46E+006	FD
30B10	68.56	6.46E+006	FD
50B06	278.16	1.12E+007	FD
50B07	269.8	1.09E+007	FD
50B08	357.13	1.05E+007	FD
50B09	347.39	1.10E+007	FD
50B10	315.46	1.10E+007	FD

Tabela 8.2: Melhores Resultados Exatos: Classe B

A Tabela 8.1 mostra os resultados obtidos para as instâncias de pequeno porte da classe C. As colunas desta tabela possuem o mesmo significado das colunas da Tabela 8.1 e da Tabela 8.1. Como estas se diferenciam da classe B apenas na quantidade de espaço em disco disponível para replicação é possível perceber claramente a influência desta restrição em disco na qualidade da função objetivo das soluções encontradas e também nos tempos computacionais. A formulação *FD* também encontrou os melhores resultados para todas as instâncias desta classe, no entanto, para duas instâncias (marcadas com *), não foi possível provar a otimalidade das soluções devido à alta demanda por memória da formulação. Nestes casos, os resultados expostos na Tabela 8.1 não são referentes à solução ótima e sim à melhor solução encontrada durante a resolução da formulação *FD*.

Instância	Tempo (s)	F.O.	Método
10C11	38.22	2.10E+006	FD
10C12	9.14	2.16E+006	FD
10C13	15.14	2.15E+006	FD
10C14	30.3	2.22E+006	FD
10C15	32.93	2.24E+006	FD
20C11	666.83	4.78E+006	FD
20C12	230.75	4.42E+006	FD
20C13	317.04	4.50E+006	FD
20C14	520.43	4.67E+006	FD
20C15	175.98	4.55E+006	FD
30C11	433.04	6.70E+006	FD
30C12	211.01	6.39E+006	FD
30C13	196.27	6.61E+006	FD
30C14	157.09	6.53E+006	FD
30C15	188.9	6.52E+006	FD
50C11	523.88	1.13E+007	FD
50C12	379.95	1.09E+007	FD
50C13*	1229.67	1.15E+007	FD
50C14*	1239.84	1.18E+007	FD
50C15	666.31	1.12E+007	FD

Tabela 8.3: Melhores Resultados Exatos: Classe C

A Tabela 8.1 mostra os resultados dos métodos exatos para as instâncias de pequeno porte da classe D. As colunas desta tabela possuem o mesmo significado das colunas das tabelas anteriores. As instâncias marcadas com ** são instâncias onde os métodos exatos não conseguem encontrar a solução ótima dentro do tempo limite estabelecido (três horas). Note que para a instância 50D16 o algoritmo *SEA* apresenta o melhor resultado dentro do tempo limite estabelecido. Isto mostra que o algoritmo SEA pode apresentar bons resultados não somente para as instâncias de grande porte, mas também para instâncias de pequeno porte, ao contrário do esperado. Os tempos computacionais apresentados para as instâncias desta classe indicam que estas instâncias são mais difíceis de serem resolvidas através de abordagens exatas do que as instâncias das classes anteriores. Os resultados obtidos pela formulação *FD* e pelo algoritmo *SEA* para esta classe de instâncias motiva o estudo e construção de novas instâncias de pequeno porte onde a demanda por memória não seja o fator crítico para o uso de métodos exatos. Este estudo/construção

de novas instâncias é alvo de trabalhos futuros bem como o aprimoramento do algoritmo *SEA* para estes casos.

Instância	Tempo (s)	F.O.	Método
10D16	8133.05	8.74E+006	FD
10D17	24547.2	8.83E+006	FD
10D18	2337.77	8.74E+006	FD
10D19	3448.68	8.53E+006	FD
10D20	12493	8.62E+006	FD
20D16**	10799.6	1.81E+007	FD
20D17**	27058.8	1.73E+007	FD
20D18**	27647.4	1.77E+007	FD
20D19**	19184.1	1.71E+007	FD
20D20**	29143.2	1.74E+007	FD
30D16**	10805.4	2.66E+007	FD
30D17**	10794.1	2.60E+007	FD
30D18**	10800.9	2.54E+007	FD
30D19**	10806.2	2.69E+007	FD
30D20**	10801.6	2.76E+007	FD
50D16**	10800	2.58E+008	SEA
50D17**	10800.6	4.13E+007	FD
50D18**	10800.4	4.80E+007	FD
50D19**	10801.2	2.41E+008	FD
50D20**	10802.7	4.48E+007	FD

Tabela 8.4: Melhores Resultados Exatos: Classe D

Para as instâncias de grande porte os resultados serão apresentados separadamente a a partir deste ponto, devido ao fato de não haverem instâncias de grande porte pertencentes às classes A e B. Isto ocorre porque durante os testes computacionais, como pode ser visto mais adiante neste capítulo, foi determinado que as instâncias destas classes são de fácil resolução. Como as instâncias de grande porte foram construídas depois das análises para as instâncias de pequeno porte estarem concluídas, julgou-se desnecessário construir instâncias com mais de 100 servidores para a classes A e B.

Instância	Tempo (s)	F.O.	Método
100C01**	10801	2.38E+007	SEA
100C02**	10802	2.25E+007	SEA
100C03**	10801	2.37E+008	SEA
100C04**	10803	2.27E+007	SEA
100C05**	10800	2.26E+007	SEA
100D16**	10801	2.74E+009	SEA
100D17**	10802	2.14E+009	SEA
100D18**	10801	1.71E+009	SEA
100D19**	10803	2.10E+009	SEA
100D20**	10800	2.61E+009	SEA

Tabela 8.5: Melhores Resultados Exatos: Instâncias de Grande Porte - 100

A Tabela 8.1 mostra os melhores resultados obtidos pelos métodos exatos para as instâncias de cem servidores. As colunas desta tabela possuem os mesmos significados das tabelas anteriores. Note que o algoritmo *SEA* foi o que encontrou as melhores soluções para todas as instâncias apresentadas e que não foi possível, por razões de tempo, provar o ótimo para estas instâncias. Os resultados apresentados são referentes às melhores soluções encontradas dentro do tempo limite para todas as instâncias apresentadas na

tabela.

Instância	Tempo (s)	F.O.	Método
200C01**	10802	2001767430.22	SEA
200C02**	10801	3649953552.48	SEA
200C03**	10800	2807880112.15	SEA
200C04**	10801	3262035383.9	SEA
200C05**	10803	5043183345.58	SEA

Tabela 8.6: Melhores Resultados Exatos: Instâncias de Grande Porte - 200

A Tabela 8.1 mostra os melhores resultados obtidos pelos métodos exatos para as instâncias com duzentos servidores. Como dito anteriormente, não instâncias da classe D com mais de 100 servidores, sendo então, todas as instâncias com mais de cem servidores pertencentes à classe C. A tabela mostra que o algoritmo *SEA* foi o que apresentou as melhores soluções para estas instâncias. Note que, também para estas instâncias, não foi possível encontrar as soluções ótimas, sendo os resultados expostos referentes à melhor solução encontrada durante o execução do algoritmo.

Instância	Tempo (s)	F.O.	Método
300C01**	10803	6090938514.1	SEA
300C02**	10801	12549058148.52	SEA
300C03**	10802	15233762018.94	SEA
300C04**	10801	12657045130.27	SEA
300C05**	10800	16377484126.23	SEA

Tabela 8.7: Melhores Resultados Exatos: Instâncias de Grande Porte - 300

A Tabela 8.1 mostra os melhores resultados obtidos pelos métodos exatos para as instâncias com trezentos servidores. As mesmas constatações feitas para as instâncias com duzentos servidores são válidas para os resultados expostos nesta tabela.

Instância	Tempo (s)	F.O.	Método
400C01**	10801	1.97E+010	SEA
400C02**	10800	3.06E+010	SEA
400C03**	10802	2.90E+010	SEA
400C04**	10800	2.85E+010	SEA
400C05**	10801	2.58E+010	SEA

Tabela 8.8: Melhores Resultados Exatos: Instâncias de Grande Porte - 400

A Tabela 8.1 mostra os melhores resultados obtidos pelos métodos exatos para as instâncias com quatrocentos servidores. As mesmas constatações feitas para os resultados obtidos para as instâncias com duzentos e trezentos servidores também se aplicam aos resultados apresentados nesta tabela. É importante notar que para nenhuma das instâncias de grande porte a formulação *FD* foi capaz de encontrar os melhores resultados. Isto se deve devido à grande demanda por memória necessária para a resolução desta formulação o que impossibilita a execução da mesma para instâncias com mais de 50 servidores nos testes feitos para esta tese. Como o algoritmo *SEA* possui uma demanda por memória menor que a formulação *FD*, é possível utilizá-lo em instâncias com muito mais do que

que 50 servidores, o que torna este algoritmo mais eficiente para as instâncias de grande porte. Para as instâncias de pequeno porte, a formulação FD consegue os melhores resultados tanto em termos de tempo quanto em termos de qualidade de solução para todas as instâncias exceto uma (50D16). Estes resultados mostram que o algoritmo *SEA* é uma solução viável para alguns casos e que aprimoramentos futuros podem tornar este algoritmo ainda mais efetivo.

Os resultados apresentados nesta seção são usados como referência para análise de *gaps* e tempos computacionais para as demais abordagens propostas. Embora os métodos exatos sejam uma abordagem adequada para a versão *offline* do problema eles também serão usados como parâmetro de comparação para a versão *online*. O objetivo desta abordagem é construir métodos capazes de fornecer soluções para a versão *online* em um tempo computacional razoável e que as soluções encontradas sejam de boa qualidade. Neste ponto, uma boa estratégia é utilizar as soluções fornecidas pelos métodos exatos como limite dual, tentando fazer com que as abordagens propostas para a versão *online* do problema se aproximem deste limite.

8.2 Resultados da Heurística *GHS*

Esta seção apresenta os resultados da heurística *GHS*, mostrada na Seção 6.5.1, que é uma solução de mercado proposta por provedores de RDC. A Tabela 8.9 mostra os resultados para a classe A de instâncias cujos nomes estão na primeira coluna, os tempos computacionais, em segundos, encontram-se na segunda coluna e a diferença percentual da qualidade das soluções obtidas por esta heurística, em relação à formulação *FD*, na última coluna. Note que a qualidade das soluções produzidas por *GHS* é extremamente baixa.

Um fato importante a mencionar é que esta heurística trata da versão *online* do PPRDR, onde não há conhecimento sobre o futuro, e a formulação *FD* trata da versão *offline*, onde todo o conhecimento futuro está disponível *a priori*, possibilitando um posicionamento de réplicas muito mais eficiente.

A Tabela 8.10 expõe os resultados para a classe B de instâncias. As colunas possuem o mesmo significado que as colunas da Tabela 8.9. Pode-se observar que a qualidade das soluções produzidas por esta heurística para esta classe é baixa, assim como nas instâncias da classe A.

A Tabela 8.11 expõe os resultados para a classe C de instâncias. As colunas possuem

Instância	Tempo (S)	Gap (%)
10A01	0.02	1850.04
10A02	0.02	2425.91
10A03	0.02	2574.09
10A04	0.02	2241.59
10A05	0.01	2066.12
20A01	0.02	2684.68
20A02	0.07	2955.16
20A03	0.05	2432.87
20A04	0.04	3286.05
20A05	0.04	2577.13
30A01	0.1	2853.89
30A02	0.07	2416.52
30A03	0.08	3556.19
30A04	0.08	3374.04
30A05	0.11	2508.12
50A01	0.26	2362.26
50A02	0.22	3039.97
50A03	0.25	3232.49
50A04	0.25	1786.7
50A05	0.27	3121.5

Tabela 8.9: GHS classe A: *gaps* e tempos

Instância	Tempo (S)	Gap (%)
10B06	0.11	93826.09
10B07	0.09	82575.35
10B08	0.09	87051.04
10B09	0.08	78339.95
10B10	0.12	100554.9
20B06	0.34	126366.26
20B07	0.38	93260.53
20B08	0.37	118577.54
20B09	0.36	110992.46
20B10	0.37	87935.59
30B06	0.69	116612.5
30B07	0.63	107781.69
30B08	0.67	108199.81
30B09	0.66	107814.73
30B10	0.6	100228.88
50B06	1.61	122487.49
50B07	1.46	114654.17
50B08	1.44	150602.51
50B09	1.56	157021.4
50B10	1.46	132586.73

Tabela 8.10: GHS classe B: *gaps* e tempos

Instância	Tempo (S)	Gap (%)
10C11	0.11	779844.99
10C12	0.12	146672.12
10C13	0.13	537058.21
10C14	0.16	907216.61
10C15	0.15	1045248.81
20C11	0.54	1752134.59
20C12	0.42	464511.29
20C13	0.46	1139118.87
20C14	0.46	1043782.47
20C15	0.4	310409.21
30C11	0.8	667360.11
30C12	0.7	248186.47
30C13	0.73	367721.55
30C14	0.72	360023.79
30C15	0.74	223162.82
50C11	1.92	466097.12
50C12	1.6	182887.5
50C13	1.98	1792017.79
50C14	2.12	1652264.86
50C15	1.76	586245.7

Tabela 8.11: GHS classe C: *gaps* e tempos

o mesmo significado que as colunas da tabelas 8.9 e 8.10. Também para esta classe a heurística GHS produz soluções de baixa qualidade quando comparadas às soluções fornecidas pela formulação *FD*.

Instância	Tempo (s)	Gap(%)
10D16	0.15	2152635.05
10D17	0.11	2036512.91
10D18	0.13	3083620.74
10D19	0.15	2975849.34
10D20	0.14	3370926.88
20D16	0.42	2900741.57
20D17	0.5	3832085.16
20D18	0.46	3868230.43
20D19	0.42	2863922
20D20	0.48	3823970.09
30D16	0.86	3790577.95
30D17	0.74	3094490.33
30D18	0.83	4233023.76
30D19	0.93	5270872.28
30D20	0.77	3297632.47
50D16	1.74	714887.1
50D17	2.05	4447336.81
50D18	2.1	5670827.92
50D19	2.01	905132.23
50D20	2.33	5477278.52

Tabela 8.12: GHS classe D: *gaps* e tempos

A Tabela 8.12 expõe os resultados para a classe D de instâncias. As colunas possuem o mesmo significado que as colunas da tabelas anteriores. Também para esta classe a heurística GHS produz soluções de baixa qualidade quando comparadas às soluções fornecidas pela formulação *FD*. Este fato ocorre em grande parte pelo fato desta heurística não permitir o encaminhamento de requisições, o que implica em muitos *backlogs* de requisições, que por sua vez são extremamente penalizados na função objetivo do problema. Por outro lado, os tempos computacionais desta heurística são muito inferiores aos obtidos

pela formulação *FD*. Isto ocorre devido à simplicidade dos algoritmos usados na estratégia adotada por esta heurística.

8.3 Resultados da Heurística *OGHS*

Esta seção apresenta os resultados obtidos pela heurística *OGHS* , que é uma versão otimizada da heurística *GHS* como descrito na Seção 6.5.1. Esta heurística é proposta com o objetivo de fazer uma comparação justa entre as heurísticas propostas e a solução de mercado.

Instância	Tempo (S)	Gap (%)
10A01	0.08	1.47
10A02	0.07	3.53
10A03	0.07	2.82
10A04	0.06	2.37
10A05	0.05	1.29
20A01	0.14	2.87
20A02	0.3	0.19
20A03	0.27	1.39
20A04	0.26	2.31
20A05	0.27	1.07
30A01	0.58	0.64
30A02	0.44	2.73
30A03	0.69	1.78
30A04	0.54	2.8
30A05	0.64	1.65
50A01	1.75	0.49
50A02	1.47	2.9
50A03	1.64	1.69
50A04	2.17	6.88
50A05	1.92	1.77

Tabela 8.13: OGHS classe A: *gaps* e tempos

A Tabela 8.13 mostra os resultados obtidos pela heurística *OGHS* para instâncias da classe A e suas colunas estão organizadas da seguinte maneira: na primeira coluna estão as instâncias de teste. A segunda coluna apresenta os tempos computacionais em segundos. A última coluna apresenta a diferença percentual entre as soluções obtidas pela heurística *OGHS* e as obtidas por *FD*. Note que os *gaps* apresentam uma redução considerável quando comparados com os obtidos pela heurística *GHS* e o gap médio de *OGHS* para estas instâncias é cerca de mil e trezentas vezes menor que o gap médio de *GHS*. Em termos de tempos computacionais, o tempo médio de *OGHS* é seis vezes maior que o de *GHS*.

A Tabela 8.14 mostra os resultados obtidos pela heurística *OGHS* para instâncias da classe B e tem as colunas organizadas do mesmo modo que a Tabela 8.13. Assim como nas instâncias da classe A, *OGHS* obteve resultados muito superiores em termos de qualidade de solução. Os tempos computacionais, assim como na classe A, sofreram aumento. A

Instância	Tempo (S)	Gap (%)
10B06	0.27	4.52
10B07	0.28	4.18
10B08	0.23	5.16
10B09	0.24	4.11
10B10	0.23	6.53
20B06	0.97	5.99
20B07	1.05	3.7
20B08	0.92	4.44
20B09	0.95	5.05
20B10	0.94	3.9
30B06	2.13	4.6
30B07	1.98	4.56
30B08	2.15	4.51
30B09	2.02	3.71
30B10	2	4.02
50B06	6.27	3.75
50B07	7.3	4.6
50B08	5.58	5.61
50B09	7.56	5.72
50B10	5.47	4.97

Tabela 8.14: OGHS classe B: *gaps* e tempos

redução no *gap* médio é da casa de de vinte e sete mil vezes, enquanto o aumento no tempo computacional médio é da casa de sete vezes.

Instância	Tempo (S)	Gap (%)
10C11	0.26	26.44
10C12	0.26	8.31
10C13	0.23	27.33
10C14	0.23	34.72
10C15	0.26	41.94
20C11	0.97	50.18
20C12	1.04	15.96
20C13	0.93	35.07
20C14	0.97	34.94
20C15	0.91	14.52
30C11	2.13	22.21
30C12	2.04	10.49
30C13	2.07	11.06
30C14	2.05	11.14
30C15	1.98	10.47
50C11	6.22	12.39
50C12	5.62	7.9
50C13	5.59	40.41
50C14	5.74	41.5
50C15	5.37	19.59

Tabela 8.15: OGHS classe C: *gaps* e tempos

A Tabela 8.15 mostra os resultados obtidos pela heurística *OGHS* para instâncias da classe C e tem as colunas organizadas do mesmo modo que as tabelas anteriores desta seção. Para esta instâncias o gap médio de *OGHS* é mais de trinta mil vezes menor que o gap médio de *GHS*. Em termos de tempo, o tempo médio de *OGHS* é cerca de três vezes maior que o de *GHS*.

A Tabela 8.16 mostra os resultados de OGHS para as instâncias de pequeno porte da classe D. Apesar de apresentar gaps muito altos para várias instâncias, os gaps para algumas delas, como por exemplo a instância 50D16, são razoáveis quando comparados

Instância	Tempo (s)	Gap(%)
10D16	0.33	17.42
10D17	0.31	15.92
10D18	0.31	27.36
10D19	0.32	7411.85
10D20	0.31	2777.93
20D16	1.08	16.29
20D17	1.22	31340.18
20D18	1.5	12871.05
20D19	1.34	16.45
20D20	1.14	7244.71
30D16	2.36	5136.84
30D17	2.22	16.44
30D18	2.42	16896.12
30D19	2.82	72303.76
30D20	2.42	15.67
50D16	6.82	2.07
50D17	8.39	7651.59
50D18	10.33	91656.42
50D19	7.84	13353.1
50D20	7.96	94794.98

Tabela 8.16: OGHS classe D: *gaps* e tempos

com os da formulação *FD*. Quando comparada com a heurística *GHS*, *OGHS* apresenta resultados melhores para todas as instâncias sendo o *gap* médio duas ordens de grandeza menor que o apresentado pelo algoritmo *GHS*. Em termos de tempo, *GHS* é da ordem de quatro vezes mais rápido que *OGHS*.

Pode-se perceber que os *gaps* de *OGHS* são muito menores que os obtidos por *GHS*. Esta redução se deve à duas características presentes na estratégia de tratamento do PDR da heurística *OGHS*: i) a possibilidade de que as requisições sejam encaminhadas para servidores diferentes do servidor de origem e ii) a possibilidade de que múltiplos servidores possam atender uma requisição ao mesmo tempo. Estas duas característica permitem uma drástica redução no número de *backlogs* feitos e consequentemente uma drástica redução na função objetivo do problema. Em termos de tempos computacionais, *OGHS* obteve resultados piores que os apresentados pela heurística *GHS*, mas melhores do que a formulação *FD*.

8.4 Resultados da heurística *HC*

Esta seção apresenta os resultados obtidos pela heurística *HC* usando os diversos estimadores apresentados na Seção 6.3. As tabelas desta seção estão organizadas da seguinte maneira: Na primeira coluna estão as instâncias. Os tempos computacionais, em segundos, encontram-se na coluna do meio. A terceira e última coluna mostra a diferença percentual da qualidade das soluções obtidas *HC*, em relação à formulação *FD*.

A Tabela 8.17 mostra os resultados de *HC* para as instâncias da classe A usando o

Instância	Tempo (s)	Gap(%)
10A01	0.07	0
10A02	0.07	0
10A03	0.06	0
10A04	0.07	0
10A05	0.06	0
20A01	0.1	0
20A02	0.31	0
20A03	0.33	0
20A04	0.24	0
20A05	0.26	0
50A01	1.52	0
50A02	1.7	0
50A03	1.48	0
50A04	1.91	6.47
50A05	1.77	0

Tabela 8.17: HC-Futuro classe A: Gaps e tempos

estimador de *conhecimento futuro*, apresentado na Seção 6.3. Esta versão de *HC* será referenciada, a partir deste ponto como *HC-Futuro*. Apesar de não ser um estimador válido, o uso deste mecanismo ainda sim é importante para obtenção de limites. Pela tabela é possível perceber que não é necessário ter o conhecimento de todos os períodos a frente para se obter as soluções ótimas para todas as instâncias desta classe exceto a instância $50A04$. Em relação aos tempos computacionais, pode-se afirmar que esta versão de HC apresenta tempos compatíveis com a heurística *OGHS* uma vez que, dados os respectivos tempos médios e desvios, os tempos se encontram estatisticamente empatados. Estes fato mostra que o uso de uma estratégia de replicação mais elaborada não tem tanto impacto no tempo computacional. Isto ocorre porque a maior parte do tempo computacional gasto por ambas as heurísticas (*OGHS* e *HC*) é gasto na resolução da formulação *RF*.

A Tabela 8.18 mostra os resultados de *HC-Futuro* para as instâncias da classe B. Pela tabela é possível observar que apenas com o conhecimento das demandas de um período adiante é possível obter as soluções ótimas para todas as instâncias desta classe. Em relação aos tempos computacionais, pode-se afirmar que *HC-Futuro* leva vantagem em relação à *OGHS* visto que ela possui tempo médio e desvios menores. Entretanto, dados os respectivos tempos médios e desvios, também para esta classe de instâncias os tempos se encontram estatisticamente empatados.

A Tabela 8.19 mostra os resultados de *HC-Futuro* para as instâncias da classe C. As colunas desta tabela possuem o mesmo significado que as colunas das demais tabelas desta seção. Pela tabela percebe-se que o conhecimento das demandas de apenas um futuro a frente não é suficiente para encontrar as soluções ótimas para a classe C usando esta heurística. No que tange os tempos computacionais, também podemos afirmar que os tempos de HC-Futuro e OGHS estão estatisticamente empatados dados os respectivos

Instância	Tempo (s)	Gap(%)
10B06	0.22	0
10B07	0.23	0
10B08	0.25	0
10B09	0.22	0
10B10	0.19	0
20B06	0.76	0
20B07	0.81	0
20B08	0.75	0
20B09	0.79	0
20B10	0.7	0
30A01	0.55	0
30A02	0.4	0
30A03	0.47	0
30A04	0.46	0
30A05	0.56	0
30B06	1.74	0
30B07	1.6	0
30B08	1.75	0
30B09	1.72	0
30B10	1.6	0
50B06	5.29	0
50B07	4.69	0
50B08	4.72	0
50B09	4.9	0
50B10	4.52	0

Tabela 8.18: HC-Futuro classe B: Gaps e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.21	0.97
10C12	0.23	0.35
10C13	0.21	1.47
10C14	0.21	2.09
10C15	0.2	3.07
20C11	0.78	3.59
20C12	0.8	0.5
20C13	0.73	1.41
20C14	0.78	1.6
20C15	0.76	0.44
30C11	1.64	0.64
30C12	1.66	0.21
30C13	1.68	0.34
30C14	1.64	0.27
30C15	1.58	0.2
50C11	5.13	0.38
50C12	4.68	0.13
50C13	4.65	2.11
50C14	4.82	3.05
50C15	4.64	0.67

Tabela 8.19: HC-Futuro classe C: Gaps e tempos

tempos médios e desvios.

Instância	Tempo (s)	Gap(%)
10D16	0.32	3.79
10D17	0.30	2.47
10D18	0.25	3.07
10D19	0.31	3.22
10D20	0.26	3.40
20D16	0.95	2.99
20D17	0.95	1.64
20D18	1.20	2.43
20D19	0.88	3.66
20D20	0.95	2.33
30D16	1.98	2.15
30D17	2.02	3.05
30D18	1.98	2.37
30D19	2.12	1.65
30D20	2.02	1.88
50D16	5.93	-0.19
50D17	5.57	1.43
50D18	7.93	2.12
50D19	6.15	2.38
50D20	6.27	1.50

Tabela 8.20: HC-Futuro classe D: Gaps e tempos

A Tabela 8.20 mostra os resultados de *HC-Futuro* para as instâncias da classe D. As colunas desta tabela possuem o mesmo significado que as colunas das demais tabelas desta seção. Pela tabela percebe-se que o conhecimento futuro não é suficiente para atingir as melhores soluções conhecidas para todas as instâncias exceto a instância 50D16. Note que para esta instância o gap apresenta um valor negativo, o que significa que a heurística *HC-Futuro* foi capaz de produzir uma solução de melhor qualidade que a apresentada pelo método exato para esta instância. Uma vez que a solução produzida pelo algoritmo *SEA* para esta instância não é um ótimo provado, este resultado é perfeitamente válido. Assim como para a classe C de instâncias, podemos afirmar que HC-Futuro e OGHS estão empatadas em termos de tempos computacionais para as instâncias da classe D.

As tabelas 8.21,8.22, 8.23 e 8.24 apresentam os resultados obtidos por *HC* usando o estimador de *Médias*, também apresentado na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Os resultados apresentados por esta versão de *HC*, referenciada a partir deste ponto como *HC-Média*, podem ser comparados com os obtidos pela heurística *OGHS*, tanto em termos de tempo computacional como em termos de qualidade de solução. Isto ocorre porque o estimador de médias é um estimador válido e portanto os resultados produzidos por qualquer heurística que use este estimador podem ser comparados com os resultados de técnicas que não usem estimadores ou ainda utilizem outros estimadores da literatura.

O resultado da comparação entre *OGHS* e *HC-Média* revela que os resultados obtidos por *HC-Média* são, em grande parte das vezes, superiores aos resultados obtidos por

Instância	Tempo (s)	Gap(%)
10A01	0.06	1.47
10A02	0.05	3.53
10A03	0.07	2.82
10A04	0.06	2.51
10A05	0.04	1.29
20A01	0.14	2.87
20A02	0.28	0.2
20A03	0.26	1.58
20A04	0.23	2.31
20A05	0.24	1.19
30A01	0.58	0.64
30A02	0.38	2.73
30A03	0.46	1.78
30A04	0.51	2.81
30A05	0.58	1.67
50A01	1.6	0.55
50A02	1.69	2.92
50A03	1.47	1.7
50A04	1.44	6.95
50A05	1.72	1.77

Tabela 8.21: HC-Médias classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.23	4.52
10B07	0.23	4.18
10B08	0.21	5.16
10B09	0.22	4.12
10B10	0.18	6.53
20B06	1	5.99
20B07	0.8	3.7
20B08	0.74	4.44
20B09	0.76	5.06
20B10	0.73	3.9
30B06	1.71	4.6
30B07	1.58	4.56
30B08	1.72	4.51
30B09	1.68	3.71
30B10	2.12	4.02
50B06	5.17	3.75
50B07	6.14	4.6
50B08	4.7	5.61
50B09	4.76	5.72
50B10	4.52	4.97

Tabela 8.22: HC-Médias classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.24	10.46
10C12	0.22	5.27
10C13	0.2	9.25
10C14	0.23	12.29
10C15	0.22	20.82
20C11	0.78	20.05
20C12	1.05	6.14
20C13	0.76	10.95
20C14	0.76	11.29
20C15	0.7	6.06
30C11	1.77	8.13
30C12	1.61	5.93
30C13	1.69	6.42
30C14	1.65	5.1
30C15	1.61	5.62
50C11	6.71	5.49
50C12	4.63	5.49
50C13	4.6	14.88
50C14	4.81	16.71
50C15	4.55	8.9

Tabela 8.23: HC-Médias classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.22	9.46
10D17	0.28	9.34
10D18	0.26	6.87
10D19	0.28	7.37
10D20	0.27	10.90
20D16	0.94	8.31
20D17	0.98	6.65
20D18	1.22	7.93
20D19	0.95	10.13
20D20	0.96	7.94
30D16	2.38	8.24
30D17	1.98	8.26
30D18	1.98	29.30
30D19	2.18	7.43
30D20	2.13	8.81
50D16	6.14	11.46
50D17	5.56	6.34
50D18	6.22	697.88
50D19	7.96	3.07
50D20	6.46	6.17

Tabela 8.24: HC-Médias classe D: *gaps* e tempos

OGHS. Na grande maioria dos casos em que *OGHS* foi melhor (9 de 10 casos) a diferença entre as duas heurísticas foi menor que $0,5\%$. A exceção é a instância 50D16 onde a heurística OGHS apresentou um resultado bem melhor que a heurística *HC-Média*. Nos casos em que *HC-Média* foi melhor (29 casos), a diferença chega a mais de 94000%. Ao analisar os resultados obtidos por classes de instâncias, percebe-se que as duas heurísticas tem o mesmo desempenho para instâncias da classe B. Também pode ser percebido que em todas as instâncias da classe C e praticamente todas as instâncias da classe D *HC-Média* obteve um desempenho melhor que *OGHS*e que, em algumas instâncias das classes A e D, *OGHS* obteve resultados melhores. Em relação aos tempos computacionais, *HC-Média* e *OGHS* são compatíveis. Estes resultados mostram que o uso de estimadores juntamente com o uso de uma estratégia de posicionamento de réplicas mais elaborada podem trazer ganhos significativos para provedores de RDC.

Instância	Tempo (s)	Gap(%)
10A01	0.07	1.47
10A02	0.06	3.63
10A03	0.08	3.11
10A04	0.08	2.53
10A05	0.03	1.41
20A01	0.09	2.87
20A02	0.34	0.25
20A03	0.26	1.55
20A04	0.24	2.38
20A05	0.24	1.18
30A01	0.58	0.71
30A02	0.37	2.76
30A03	0.44	1.84
30A04	0.51	2.84
30A05	0.6	1.71
50A01	1.57	0.59
50A02	1.28	3.01
50A03	1.45	1.7
50A04	1.46	6.99
50A05	1.78	1.77

Tabela 8.25: HC-AELB classe A: *gaps* e tempos

As tabelas 8.25, 8.26, 8.27 e 8.28 mostram os resultados de *HC* usando o estimador *AELB*, mostrado na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Ao comparar-se esta versão, referenciada a partir deste ponto como *HC-AELB*, com a heurística *OGHS* pode-se perceber que elas tem o mesmo desempenho para instâncias da classe B, *OGHS* é ligeiramente melhor para muitas instâncias da classe A e *HC-AELB* é melhor em todas as instâncias das classes C e D. Ao comparar-se *HC-AELB* com *HC-Média*, nota-se que o estimador *AELB* produz resultados melhores para todas as instâncias das classes C e D e para algumas instâncias da classe A. Entretanto, para a maioria das instâncias da classe A, o estimador *AELB* apresentou um resultado pior que o estimador de médias. Para as instâncias da classe B as duas versões da heurística apresentam o mesmo resultado.

Instância	Tempo (s)	Gap(%)
10B06	0.22	4.52
10B07	0.21	4.18
10B08	0.23	5.16
10B09	0.2	4.12
10B10	0.18	6.53
20B06	0.74	5.99
20B07	0.82	3.7
20B08	0.75	4.44
20B09	0.74	5.06
20B10	0.75	3.9
30B06	1.76	4.6
30B07	1.6	4.56
30B08	1.68	4.51
30B09	1.68	3.71
30B10	1.6	4.02
50B06	5.12	3.75
50B07	4.65	4.6
50B08	4.62	5.61
50B09	6.12	5.72
50B10	5.72	4.97

Tabela 8.26: HC-AELB classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.21	6.06
10C12	0.22	4.55
10C13	0.18	6.66
10C14	0.2	7.76
10C15	0.23	10.93
20C11	0.74	9
20C12	0.82	4.51
20C13	0.76	6.49
20C14	0.73	7.53
20C15	0.78	4.64
30C11	1.7	5.31
30C12	1.62	4.95
30C13	1.72	5.2
30C14	1.68	4.09
30C15	1.62	4.45
50C11	5.2	4.31
50C12	4.7	4.84
50C13	4.58	8.24
50C14	4.83	9.09
50C15	4.57	6.14

Tabela 8.27: HC-AELB classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.24	5.19
10D17	0.24	4.52
10D18	0.25	4.58
10D19	0.25	3.97
10D20	0.23	4.98
20D16	0.90	4.89
20D17	0.93	2.98
20D18	1.18	4.00
20D19	0.89	4.58
20D20	0.91	3.73
30D16	1.98	3.38
30D17	1.93	4.20
30D18	1.94	3.80
30D19	2.15	2.84
30D20	2.15	4.24
50D16	6.05	0.06
50D17	5.5	2.86
50D18	6.14	3.61
50D19	6.14	2.63
50D20	6.13	2.47

Tabela 8.28: HC-AELB classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.07	1.47
10A02	0.07	3.53
10A03	0.05	2.82
10A04	0.07	2.37
10A05	0.05	1.29
20A01	0.11	2.87
20A02	0.32	0.19
20A03	0.26	1.39
20A04	0.23	2.31
20A05	0.24	1.07
30A01	0.54	0.64
30A02	0.43	2.73
30A03	0.51	1.78
30A04	0.5	2.8
30A05	0.56	1.65
50A01	1.56	0.49
50A02	1.31	2.9
50A03	1.45	1.7
50A04	1.92	6.88
50A05	1.76	1.77

Tabela 8.29: HC-AEBH classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.22	4.52
10B07	0.21	4.18
10B08	0.18	5.16
10B09	0.22	4.12
10B10	0.21	6.53
20B06	0.74	5.99
20B07	0.77	3.7
20B08	0.78	4.44
20B09	0.73	5.06
20B10	0.75	3.9
30B06	2.36	4.6
30B07	1.66	4.56
30B08	1.69	4.51
30B09	1.9	3.71
30B10	1.64	4.02
50B06	5.27	3.75
50B07	4.7	4.6
50B08	4.78	5.61
50B09	4.8	5.72
50B10	4.58	4.97

Tabela 8.30: HC-AEBH classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.23	7.68
10C12	0.21	4.46
10C13	0.2	9.51
10C14	0.21	9.58
10C15	0.19	12.29
20C11	0.71	12.46
20C12	0.85	5.37
20C13	0.73	8.37
20C14	0.77	9.14
20C15	0.74	5.42
30C11	1.74	6.54
30C12	2.22	5.33
30C13	1.74	5.17
30C14	1.71	4.66
30C15	1.56	5
50C11	5.21	5.01
50C12	4.69	5
50C13	4.68	11.18
50C14	4.79	11.07
50C15	5.99	7.06

Tabela 8.31: HC-AEBH classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.25	4.49
10D17	0.25	4.74
10D18	0.24	4.26
10D19	0.26	3.67
10D20	0.26	5.08
20D16	0.91	4.23
20D17	0.93	2.83
20D18	0.92	4.09
20D19	0.90	4.44
20D20	1.16	3.58
30D16	2.00	2.88
30D17	1.95	4.06
30D18	1.94	3.07
30D19	2.11	2.58
30D20	2.07	3.14
50D16	5.86	-0.11
50D17	5.56	2.88
50D18	6.10	2.92
50D19	6.10	2.64
50D20	6.83	2.56

Tabela 8.32: HC-AEBH classe D: *gaps* e tempos

As tabelas 8.29, 8.30, 8.31 e 8.32 mostram os resultados obtidos por *HC* usando o estimador *AEBH*, mostrado na Seção 6.3. Ao comparar-se os resultados desta versão, chamada de *HC-AEBH*, com a heurística *OGHS*, percebe-se que as duas heurísticas apresentam o mesmo resultado para a classe B, *HC-AEBH* apresenta resultados muito próximos aos de *OGHS* para a classe A e melhores para as classe C e D. Comparando com as versões *HC-Média* e *HC-AELB* pode-se observar que as três versões apresentam resultados idênticos para a classe B de instâncias. A versão *HC-AEBH* apresenta resultados melhores ou iguais aos da versão *HC-Média*, exceto para a instância *50A03*. Em relação à versão *HC-AELB*, a versão *HC-AEBH* apresenta resultados melhores ou iguais para todas as instâncias da classe A exceto pela instância *50A03*. Já para as classes C e D, *HC-AEBH* apresenta resultados melhores para a maioria das instâncias. Em termos de tempo computacional, As três versões apresentam resultados semelhantes visto que o tempo dos estimadores é praticamente desprezível e que as três versões usam os mesmos mecanismos para resolver o PDR e o PPR.

Um outro resultado interessante a ser observado é que o estimador *AEBH* tende a apresentar resultados melhores que o estimador *AELB*, apesar dos dois estimadores serem baseados na técnica de alisamento exponencial. Em 66 dos 80 casos, o estimador *AEBH* apresentou resultados iguais ou mais precisos que o estimador *AELB*. Uma possível explicação para isso é que o estimador *AEBH* possui duas constantes de alisamento, tornando este estimador mais flexível do que o estimador *AELB* que possuí apenas uma constante.

As tabelas 8.33, 8.34, 8.35 e 8.36 mostram os resultados de *HC* usando o estimador

Instância	Tempo (s)	Gap(%)
10A01	0.06	1.47
10A02	0.06	3.53
10A03	0.07	2.82
10A04	0.05	2.37
10A05	0.04	1.29
20A01	0.14	2.87
20A02	0.32	0.19
20A03	0.28	1.39
20A04	0.22	2.31
20A05	0.23	1.07
30A01	0.51	0.64
30A02	0.4	2.73
30A03	0.49	1.78
30A04	0.48	2.8
30A05	0.62	1.65
50A01	1.62	0.49
50A02	1.27	2.9
50A03	1.42	1.7
50A04	1.47	6.88
50A05	1.73	1.77

Tabela 8.33: HC-Bote classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.19	4.52
10B07	0.23	4.18
10B08	0.2	5.16
10B09	0.19	4.12
10B10	0.23	6.53
20B06	0.77	5.99
20B07	0.85	3.7
20B08	0.75	4.44
20B09	0.79	5.06
20B10	0.74	3.9
30B06	1.76	4.6
30B07	1.6	4.56
30B08	1.75	4.51
30B09	1.67	3.71
30B10	1.92	4.02
50B06	5.18	3.75
50B07	5.93	4.6
50B08	6	5.61
50B09	4.87	5.72
50B10	4.58	4.97

Tabela 8.34: HC-Bote classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.23	6.27
10C12	0.2	4.26
10C13	0.22	7.4
10C14	0.22	7.63
10C15	0.21	10.86
20C11	0.74	9.67
20C12	0.82	4.5
20C13	0.74	6.97
20C14	0.76	7.47
20C15	0.74	4.82
30C11	1.71	5.5
30C12	1.57	4.94
30C13	1.7	5.21
30C14	1.7	4.15
30C15	1.57	4.38
50C11	5.9	4.37
50C12	4.68	4.78
50C13	4.6	8.69
50C14	4.73	9.12
50C15	4.54	6.22

Tabela 8.35: HC-Bote classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.27	5.00
10D17	0.26	4.52
10D18	0.25	4.68
10D19	0.25	3.88
10D20	0.27	5.52
20D16	1.26	4.14
20D17	0.96	2.89
20D18	1.25	4.07
20D19	0.90	4.64
20D20	1.25	3.65
30D16	1.96	3.07
30D17	2.04	4.42
30D18	1.97	3.34
30D19	2.18	2.96
30D20	2.08	3.20
50D16	5.99	-0.05
50D17	7.18	3.07
50D18	6.10	3.08
50D19	7.90	2.68
50D20	8.26	2.74

Tabela 8.36: HC-Bote classe D: *gaps* e tempos

Bote, também apresentado na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Os resultados desta versão, referenciada a partir de agora como *HC-Bote*, são importantes devido ao fato de esta heurística apresentar resultados iguais ou melhores que os da versão *HC-Média* e que os da heurística *OGHS* em termos de qualidade de solução. Como os tempos computacionais das três heurísticas são compatíveis e os tempos médio, mais curto e mais longo de *HC-Bote* são inferiores aos respectivos tempos de *OGHS*, pode-se concluir que *HC-Bote* supera *OGHS* e *HC-Média* pois produz resultados de melhor qualidade em um tempo computacional compatível.

Ao comparar a versão *HC-Bote* com as versões *HC-AELB* e *HC-AEBH*, pode ser observado que as três versões apresentam os mesmos resultados para a classe B de instâncias. Comparando as versões *HC-Bote* com *HC-AELB*, constata-se que *HC-Bote* apresenta resultados iguais ou melhores para todas as instâncias da classe A. Para a classe C *HC-Bote* apresenta resultados melhores para 13 das 20 instâncias. Para as instâncias da classe D, *HC-Bote* apresenta resultados melhores para 11 das 20 instâncias. Comparando as versões *HC-Bote* e *HC-AEBH*, percebe-se que a primeira é igual ou melhor à última para as instâncias da classe A. Já para as classes C e D a maior vantagem é de *HC-AEBH*, sendo esta última melhor em 15 dos 20 casos da classe C e em 17 dos 20 casos da classe D. Estes resultados mostram que apesar de sua simplicidade, o estimador *Bote* apresenta resultados tão bons quanto os demais estimadores para as instâncias analisadas, visto que a maior desvantagem deste estimador foi de $1,07\%$, obtido pelo estimador *AEBH* para a instância $10C15$, e a maior vantagem foi de $10,38\%$, obtida sobre estimador de médias para a instância $20C11$. Em relação à heurística *OGHS*, que não utiliza estimadores, a maior vantagem foi de $94792,24\%$ para a instância $50D20$.

8.5 Resultados da Heurística HCAGAP

Esta seção apresenta os resultados obtidos pela heurística *HCAGAP*, exposta na Seção 6.5.3, usando os diversos estimadores apresentados na Seção 6.3. As tabelas desta seção seguem a seguinte organização: Na coluna um são mostradas as instâncias de teste. Na coluna dois estão os tempos computacionais, expressos em segundos E na coluna 3 estão as diferenças percentuais dos valores das funções objetivo entre as soluções obtidas *HCAGAP* e as soluções obtidas por *FD*.

As tabelas 8.37, 8.38, 8.39 e 8.40 apresentam os resultados da heurística *HCAGAP* com o conhecimento futuro. A principal análise que pode ser feita com estes resultados é

Instância	Tempo (s)	Gap(%)
10A01	0.08	0
10A02	0.06	0
10A03	0.09	0
10A04	0.06	0
10A05	0.04	0
20A01	0.1	0
20A02	0.31	0
20A03	0.26	0
20A04	0.22	0
20A05	0.22	0
30A01	0.59	0
30A02	0.4	0
30A03	0.49	0
30A04	0.46	0
30A05	0.57	0
50A01	1.63	0
50A02	1.37	0
50A03	1.53	0
50A04	1.52	6.47
50A05	2.13	0

Tabela 8.37: HCAGAP-Futuro classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.24	0
10B07	0.25	0
10B08	0.2	0
10B09	0.19	0
10B10	0.21	0
20B06	0.78	0
20B07	0.86	0
20B08	0.96	0
20B09	0.8	0
20B10	0.72	0
30B06	1.8	0
30B07	1.66	0
30B08	1.79	0
30B09	1.77	0
30B10	1.63	0
50B06	5.4	0
50B07	4.9	0
50B08	4.92	0
50B09	5.1	0
50B10	4.75	0

Tabela 8.38: HCAGAP-Futuro classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.23	0.96
10C12	0.2	0.35
10C13	0.24	1.25
10C14	0.24	1.53
10C15	0.26	1.86
20C11	0.84	3.28
20C12	0.82	0.47
20C13	0.83	1.29
20C14	0.86	1.3
20C15	0.78	0.48
30C11	1.88	0.62
30C12	1.68	0.21
30C13	1.76	0.33
30C14	1.8	0.27
30C15	1.6	0.2
50C11	5.5	0.35
50C12	5.06	0.13
50C13	5.22	1.91
50C14	6.23	1.99
50C15	6.33	0.61

Tabela 8.39: HCAGAP-Futuro classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.41	2.94
10D17	0.44	2.61
10D18	0.38	3.10
10D19	0.38	3.20
10D20	0.46	3.14
20D16	1.64	2.40
20D17	1.44	1.70
20D18	1.48	2.42
20D19	2.03	2.75
20D20	1.45	1.83
30D16	3.46	1.93
30D17	4.82	2.79
30D18	3.67	2.04
30D19	3.49	1.28
30D20	3.36	1.99
50D16	9.36	0
50D17	10.58	1.15
50D18	9.63	1.89
50D19	9.08	2.69
50D20	9.89	1.13

Tabela 8.40: HCAGAP-Futuro classe D: *gaps* e tempos

a respeito da qualidade das soluções produzidas pela heurística *GAS* de posicionamento de réplicas. A versão *HC-Futuro*, pode ser comparada com esta versão de *HCAGAP*, chamada a partir deste ponto de *HCAGAP-Futuro*. Esta comparação pode ser feita devido ao fato de que estas duas versões possuem dois de seus três componentes idênticos, a saber, a formulação *RF* para a resolução do PDR e o conhecimento futuro como estimador. A diferença entre estas heurísticas está no componente usado na resolução do PPR, que na versão *HC-Futuro* é a heurística *GAS* e nesta versão de *HCAGAP*, chamada de *HCAGAP-Futuro*, é a formulação *AGAP*. É importante ressaltar que apesar de ser um algoritmo guloso, a heurística *GAS* obteve resultados muito bons quando comparados com os da formulação *AGAP*. Também é importante relatar que para algumas instâncias a heurística *GAS* obteve um resultado melhor do que a formulação *AGAP*. Este resultado, que pode parecer a princípio incoerente, é devido ao fato de que a heurística *GAS* e a formulação *AGAP* podem tomar decisões diferentes. Para a instância 20C15 por exemplo, a diferença ocorre ao tentar-se inserir o conteúdo 10 no servidor 9 no período 5. Existem dois candidatos para remoção neste caso, os conteúdos 7 e 8, ambos com demanda igual dentro do horizonte conhecido. Como cada um dos métodos tem seu próprio critério de escolha para remoção, o que ocorre é que a formulação *AGAP* escolhe o conteúdo 8 para remoção enquanto a heurística *GAS* escolhe o conteúdo 7. Ao deslizar-se o horizonte para o próximo período, percebe-se que a decisão tomada pela heurística *GAS* é mais vantajosa, visto mais requisições chegam para o conteúdo 8, mantido pela heurística *GAS*, do que para o conteúdo 7. Neste novo horizonte, a formulação *AGAP* corrige o erro anterior, trazendo novamente para o servidor o conteúdo removido. Entretanto, a correção do erro implica em um novo download do conteúdo, o que também é contabilizado pela função objetivo. É possível corrigir esta diferença através de uma análise de histórico das demandas, que possivelmente revelaria qual o conteúdo mais adequado para manter. Entretanto, como isso implicaria em uma mudança na estratégia do algoritmo, optou-se por apenas explicar o ocorrido e ressaltar que tanto *HC-Futuro* como *HCAGAP-Futuro* são heurísticas e portanto estão sujeitas a este tipo de acontecimento.

Em relação aos tempos computacionais, também pode ser percebido pelas tabelas 8.37, 8.38, 8.39 e 8.40 que *HCAGAP* demanda mais tempo para completar sua execução. Isto ocorre devido ao fato de que para chegar aos seus resultados, esta heurística utiliza em cada período duas formulações matemáticas, sendo uma contínua (RF) e uma inteira ($AGAP$), enquanto *HC* utiliza apenas uma formulação contínua.

As tabelas 8.41, 8.42, 8.43 e 8.44 expõem os resultados da versão de *HCAGAP* com o estimador *Médias* (*HCAGAP-Média*) para as classes de instâncias A, B, C e D res-

Instância	Tempo (s)	Gap(%)
10A01	0.06	1.47
10A02	0.06	3.53
10A03	0.07	2.82
10A04	0.06	2.51
10A05	0.06	1.29
20A01	0.1	2.87
20A02	0.3	0.2
20A03	0.29	1.58
20A04	0.24	2.31
20A05	0.23	1.19
30A01	0.55	0.64
30A02	0.41	2.73
30A03	0.5	1.78
30A04	0.5	2.81
30A05	0.58	1.67
50A01	1.62	0.55
50A02	1.3	2.92
50A03	1.45	1.7
50A04	1.51	6.95
50A05	1.79	1.77

Tabela 8.41: HCAGAP-Médias classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.2	4.52
10B07	0.25	4.18
10B08	0.2	5.16
10B09	0.23	4.12
10B10	0.22	6.53
20B06	0.77	5.99
20B07	0.84	3.7
20B08	0.8	4.44
20B09	0.84	5.06
20B10	0.76	3.9
30B06	2.43	4.6
30B07	1.66	4.56
30B08	1.8	4.51
30B09	1.77	3.71
30B10	1.66	4.02
50B06	5.5	3.75
50B07	4.88	4.6
50B08	5	5.61
50B09	5.2	5.72
50B10	4.74	4.97

Tabela 8.42: HCAGAP-Médias classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.28	10.48
10C12	0.25	5.25
10C13	0.25	8.86
10C14	0.28	11.61
10C15	0.26	20.54
20C11	0.92	19.25
20C12	0.91	6.25
20C13	0.9	10.1
20C14	0.86	11.02
20C15	0.78	6.06
30C11	1.85	7.77
30C12	1.68	5.91
30C13	2.08	6.21
30C14	1.8	5.1
30C15	1.63	5.56
50C11	5.49	5.45
50C12	4.92	5.45
50C13	5.22	14.77
50C14	5.55	14.9
50C15	6.52	8.72

Tabela 8.43: HCAGAP-Médias classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.50	8.68
10D17	0.46	8.30
10D18	0.41	6.23
10D19	0.36	7.38
10D20	0.50	8.65
20D16	2.25	8.49
20D17	1.35	6.11
20D18	1.54	7.45
20D19	1.63	7.88
20D20	1.62	6.70
30D16	4.68	8.69
30D17	3.45	8.22
30D18	5.08	8.05
30D19	3.64	6.96
30D20	3.38	8.32
50D16	9.86	10.51
50D17	13.85	6.02
50D18	9.85	321.73
50D19	10.93	3.34
50D20	11.15	5.54

Tabela 8.44: HCAGAP-Médias classe D: *gaps* e tempos

pectivamente. Ao comparar esta versão com a versão *HC-Média*, pode-se constatar que *HCAGAP-Média* produz resultados iguais ou melhores para a maioria dos casos. Entretanto, em nenhum caso esta versão conseguiu atingir resultados melhores que os da versão *HC-Bote*. Estes resultados reforçam que a formulação *AGAP* pode produzir resultados melhores que a heurística *GAS* em termos de qualidade de solução e também mostram que para as instâncias analisadas, não é suficiente ter bons algoritmos de posicionamento de réplicas e bons mecanismos de distribuição de requisições. Observe que mesmo tendo o mesmo mecanismo de resolução para o PDR e um mecanismo que produz melhores resultados para o PPR *HCAGAP-Média* não consegue superar *HC-Bote*. Isto mostra que estimadores com boa precisão também se fazem necessários para a obtenção de bons resultados.

Instância	Tempo (s)	Gap(%)
10A01	0.07	1.47
10A02	0.06	3.59
10A03	0.07	3.11
10A04	0.08	2.59
10A05	0.06	1.39
20A01	0.11	2.87
20A02	0.33	0.23
20A03	0.27	1.51
20A04	0.25	2.36
20A05	0.25	1.17
30A01	0.58	0.66
30A02	0.46	2.75
30A03	0.64	1.84
30A04	0.48	2.83
30A05	0.61	1.71
50A01	1.6	0.54
50A02	1.36	2.95
50A03	1.52	1.7
50A04	1.57	6.95
50A05	1.78	1.77

Tabela 8.45: HCAGAP-AELB classe A: *gaps* e tempos

As tabelas 8.45, 8.46, 8.47 e 8.48 mostram os *gaps* e os tempos computacionais para a versão de *HCAGAP* que usa o estimador *AELB* (*HCAGAP-AELB*), para as instâncias das classes A, B, C e D respectivamente. Quando comparados com os resultados de *HC-AELB*, *HCAGAP-AELB* é superior ou igual em termos de qualidade de solução para todas as instâncias das classes A, B e C, mostrando novamente que, um bom mecanismo de posicionamento de réplicas tem impacto na qualidade das soluções. Para a classe D de instâncias *HCAGAP-AELB* produz resultados melhores em 16 dos 20 casos quando comparada com *HC-AELB*. Em relação à versão *HC-Bote*, *HCAGAP-AELB* obtém resultados melhores para 11 das 20 instâncias da classe C e para 12 das 20 instâncias da classe D. Nas demais instâncias da classe C e D e na maioria das instâncias da classe A *HC-Bote* obteve melhores resultados. Ambas as versões obtiveram os mesmos resultados para a classe B.

Instância	Tempo (s)	Gap(%)
10B06	0.21	4.52
10B07	0.26	4.18
10B08	0.24	5.16
10B09	0.2	4.12
10B10	0.2	6.53
20B06	0.79	5.99
20B07	0.86	3.7
20B08	0.78	4.44
20B09	0.81	5.06
20B10	0.75	3.9
30B06	1.85	4.6
30B07	1.64	4.56
30B08	2.13	4.51
30B09	1.82	3.71
30B10	1.63	4.02
50B06	5.41	3.75
50B07	4.9	4.6
50B08	4.86	5.61
50B09	5.1	5.72
50B10	4.79	4.97

Tabela 8.46: HCAGAP-AELB classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.29	6.77
10C12	0.24	4.62
10C13	0.24	6.86
10C14	0.26	7.73
10C15	0.24	10.76
20C11	0.88	8.92
20C12	0.88	4.27
20C13	0.84	6.45
20C14	0.88	7.44
20C15	0.93	4.6
30C11	1.88	5.61
30C12	1.66	5.08
30C13	1.8	4.93
30C14	1.78	4.23
30C15	2.15	4.55
50C11	5.56	4.39
50C12	4.96	4.85
50C13	6.22	8.13
50C14	6.92	8.33
50C15	5.01	6.2

Tabela 8.47: HCAGAP-AELB classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.41	4.99
10D17	0.40	4.39
10D18	0.39	4.31
10D19	0.34	3.96
10D20	0.48	5.10
20D16	1.63	4.17
20D17	1.35	2.83
20D18	1.43	3.98
20D19	1.65	3.99
20D20	1.49	3.51
30D16	3.22	3.40
30D17	3.34	3.91
30D18	3.47	3.52
30D19	3.41	2.69
30D20	3.31	3.92
50D16	9.69	0.26
50D17	11.77	2.71
50D18	9.49	3.28
50D19	11.04	2.92
50D20	9.90	2.11

Tabela 8.48: HCAGAP-AELB classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.07	1.47
10A02	0.06	3.53
10A03	0.07	2.82
10A04	0.07	2.4
10A05	0.04	1.29
20A01	0.14	2.87
20A02	0.29	0.19
20A03	0.26	1.39
20A04	0.28	2.32
20A05	0.26	1.09
30A01	0.56	0.64
30A02	0.39	2.73
30A03	0.5	1.78
30A04	0.52	2.8
30A05	0.62	1.65
50A01	1.62	0.49
50A02	1.32	2.92
50A03	1.54	1.71
50A04	1.52	6.89
50A05	1.83	1.77

Tabela 8.49: HCAGAP-AEBH classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.2	4.52
10B07	0.22	4.18
10B08	0.21	5.16
10B09	0.2	4.12
10B10	0.22	6.53
20B06	0.86	5.99
20B07	0.88	3.7
20B08	0.77	4.44
20B09	0.82	5.06
20B10	0.72	3.9
30B06	1.84	4.6
30B07	1.65	4.56
30B08	2.41	4.51
30B09	1.71	3.71
30B10	1.65	4.02
50B06	5.42	3.75
50B07	4.94	4.6
50B08	5.03	5.61
50B09	5.05	5.72
50B10	4.76	4.97

Tabela 8.50: HCAGAP-AEBH classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.23	6.12
10C12	0.21	4.24
10C13	0.24	6.15
10C14	0.24	6.86
10C15	0.25	8.83
20C11	0.88	9.28
20C12	0.9	4.25
20C13	0.8	6.38
20C14	0.92	7.25
20C15	0.81	4.66
30C11	1.9	5.39
30C12	1.71	4.89
30C13	1.87	5
30C14	1.76	4.12
30C15	1.63	4.36
50C11	5.46	4.24
50C12	4.92	4.74
50C13	6.97	8.32
50C14	5.2	7.95
50C15	4.9	6.08

Tabela 8.51: HCAGAP-AEBH classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.41	3.86
10D17	0.44	4.75
10D18	0.41	4.29
10D19	0.38	3.67
10D20	0.44	5.06
20D16	1.47	3.73
20D17	1.26	2.66
20D18	1.37	3.79
20D19	1.55	3.92
20D20	1.47	3.20
30D16	3.25	2.79
30D17	3.39	3.84
30D18	3.35	3.06
30D19	3.57	2.45
30D20	3.36	3.07
50D16	10.32	0.11
50D17	8.56	2.65
50D18	9.50	2.83
50D19	12.40	2.93
50D20	12.30	2.23

Tabela 8.52: HCAGAP-AEBH classe D: *gaps* e tempos

As tabelas 8.49, 8.50, 8.51 e 8.52 apresentam os resultados para a versão de *HCAGAP* que utiliza o estimador *AEBH* (*HCAGAP-AEBH*) para as classes A, B, C e D respectivamente. Estes resultados quando comparados com os da versão *HC-AEBH* reforçam o fato de que a formulação *AGAP* é capaz de gerar resultados melhores que os da heurística *GAS*. *HCAGAP-AEBH* apresenta resultados iguais ou de melhor qualidade que *HC-AEBH* para todas as instâncias das classes A, B e C. Para as instâncias da classe D HCAGAP-AEBH apresenta resultados melhores para 16 dos 20 casos. Comparada com *HC-Bote*, a versão *HCAGAP-AEBH* perde em 10 das 80 instâncias (pertencentes às classes A e D), e apresenta resultados melhores ou iguais nos demais casos. Quando comparada com a versão *HCAGAP-AELB*, a versão *HCAGAP-AEBH* perde em apenas 8 casos. Este resultado reforça o fato de que o estimador *AEBH* tende a apresentar resultados melhores que o estimador *AELB*.

As tabelas 8.53, 8.54, 8.55 e 8.56 mostram os resultados para a versão de *HCAGAP* que utiliza o estimador *Bote* (*HCAGAP-Bote*) para as classes A, B, C e D de instâncias respectivamente. Estes resultados quando comparados com os da versão *HC-Bote* mostram que *HCAGAP-Bote* apresenta resultados iguais ou melhores em termos de qualidade de solução para 71 das 80. O fato da heurística *GAS* apresentar resultados melhores que a formulação *AGAP* ocorre com todos os estimadores e também para o caso em que o conhecimento futuro é utilizado. Além das mesmas razões apresentadas anteriormente para o melhor desempenho para algumas instâncias da heurística *GAS* no caso do conhecimento futuro, pode-se citar outra possível causa para este acontecimento no caso dos estimadores. As duas abordagens (*GAS* e *AGAP*) possuem seus próprios critérios para remoção e

Instância	Tempo (s)	Gap(%)
10A01	0.07	1.47
10A02	0.08	3.53
10A03	0.09	2.82
10A04	0.08	2.37
10A05	0.05	1.29
20A01	0.15	2.87
20A02	0.31	0.19
20A03	0.24	1.39
20A04	0.25	2.31
20A05	0.24	1.07
30A01	0.55	0.64
30A02	0.4	2.73
30A03	0.5	1.78
30A04	0.52	2.8
30A05	0.62	1.65
50A01	2.22	0.49
50A02	1.29	2.9
50A03	1.54	1.7
50A04	1.51	6.88
50A05	1.79	1.77

Tabela 8.53: HCAGAP-Bote classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.22	4.52
10B07	0.25	4.18
10B08	0.18	5.16
10B09	0.21	4.12
10B10	0.19	6.53
20B06	0.8	5.99
20B07	0.83	3.7
20B08	0.8	4.44
20B09	0.8	5.06
20B10	0.76	3.9
30B06	1.81	4.6
30B07	1.93	4.56
30B08	2.38	4.51
30B09	2.35	3.71
30B10	1.66	4.02
50B06	5.4	3.75
50B07	4.97	4.6
50B08	4.94	5.61
50B09	5.12	5.72
50B10	4.8	4.97

Tabela 8.54: HCAGAP-Bote classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.27	6.21
10C12	0.24	4.26
10C13	0.23	6.68
10C14	0.2	7.01
10C15	0.26	9.41
20C11	0.89	9.32
20C12	0.88	4.32
20C13	1.07	6.84
20C14	0.87	7.17
20C15	0.74	4.6
30C11	1.92	5.32
30C12	1.69	4.92
30C13	1.8	5.23
30C14	1.78	4.07
30C15	2.23	4.33
50C11	5.67	4.33
50C12	4.93	4.8
50C13	5.16	8.35
50C14	6.82	8.2
50C15	4.92	6.19

Tabela 8.55: HCAGAP-Bote classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.46	4.45
10D17	0.39	4.66
10D18	0.35	4.71
10D19	0.35	4.01
10D20	0.50	5.19
20D16	2.02	3.97
20D17	1.35	2.82
20D18	1.46	4.12
20D19	1.45	4.01
20D20	1.46	3.46
30D16	3.24	2.66
30D17	3.36	4.25
30D18	3.39	3.20
30D19	3.55	2.80
30D20	3.44	3.31
50D16	9.17	0.15
50D17	8.38	2.83
50D18	9.14	2.99
50D19	9.01	2.96
50D20	9.72	2.48

Tabela 8.56: HCAGAP-Bote classe D: *gaps* e tempos

inserção de conteúdos, entretanto para tomar as decisões tanto *GAS* quanto *AGAP* utilizam os resultados das previsões das demandas. Os estimadores *Médias*, *AELB*, *AEBH* e *Bote*, ao contrário do conhecimento futuro, apresentam erros em suas previsões. Assim, ao passar previsões que na verdade não se concretizam para as abordagens de resolução do PPR, os estimadores podem afetar os resultados obtidos tanto pela formulação *AGAP* quanto pela heurística *GAS*.

Comparando os resultados de *HCAGAP-Bote* com a versão *HCAGAP-AELB*, observa-se que as duas heurísticas apresentam o mesmo resultado em termos de qualidade de solução para as instâncias da classe B. Para a classe A de instâncias *HCAGAP-Bote* apresenta resultados iguais ou melhores em termos de qualidade de solução. Também pode ser observado que para cinco instâncias da classe C *HCAGAP-AELB* apresenta soluções de melhor qualidade, sendo *HCAGAP-Bote* melhor nos outros quinze casos. Para a classe D de instâncias *HCAGAP-Bote* supera *HCAGAP-AELB* em nove casos.

Quando comparada com *HCAGAP-AEBH*, *HCAGAP-Bote* apresenta resultados ligeiramente piores (menos de 1%) em 15 das 20 instâncias da classe C e ligeiramente melhores nas demais instâncias desta classe. Para a classe D *HCAGAP-AEBH* apresenta resultados ligeiramente melhores em 18 das 20 instâncias desta classe, sendo *HCAGAP-Bote* ligeiramente melhor nas demais instâncias. Para a classe A *HCAGAP-Bote* apresenta resultados iguais ou ligeiramente melhores para todas as instâncias. As duas versões obtiveram o mesmo desempenho em instâncias da classe B. Estes resultados novamente mostram o bom desempenho do estimador *Bote* em relação aos outros para as instâncias analisadas.

8.6 Resultados da heurística *HNH*

Esta seção apresenta os resultados obtidos pela heurística *HNH* usando os diversos estimadores apresentados na Seção 6.3. As tabelas desta seção mostram as instâncias na primeira coluna, os tempos computacionais, em segundos, na segunda coluna e a diferença percentual da qualidade das soluções obtidas pela heurística, em relação à formulação *FD*, na última coluna.

As tabelas 8.57, 8.58, 8.59 e 8.60 mostram os resultados para a versão de *HNH* que utiliza o conhecimento futuro (*HNH-Futuro*) para as instâncias das classes A, B, C e D respectivamente. O grande trunfo desta heurística é fato de que ela consegue atingir resultados com exatamente a mesma qualidade da heurística *HCAGAP* com tempos mais baixos. De fato o tempo médio obtido pela heurística *HNH-Futuro* é apenas dois décimos

Instância	Tempo (s)	Gap(%)
10A01	0.04	0
10A02	0.04	0
10A03	0.05	0
10A04	0.04	0
10A05	0.03	0
20A01	0.08	0
20A02	0.16	0
20A03	0.16	0
20A04	0.13	0
20A05	0.14	0
30A01	0.28	0
30A02	0.2	0
30A03	0.26	0
30A04	0.28	0
30A05	0.33	0
50A01	0.71	0
50A02	0.58	0
50A03	0.71	0
50A04	0.86	6.47
50A05	0.77	0

Tabela 8.57: HNH-Futuro classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.16	0
10B07	0.16	0
10B08	0.14	0
10B09	0.12	0
10B10	0.12	0
20B06	0.4	0
20B07	0.46	0
20B08	0.44	0
20B09	0.42	0
20B10	0.41	0
30B06	0.84	0
30B07	0.81	0
30B08	0.86	0
30B09	0.83	0
30B10	0.82	0
50B06	2.27	0
50B07	2.1	0
50B08	2.1	0
50B09	2.18	0
50B10	2.03	0

Tabela 8.58: HNH-Futuro classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.18	0.96
10C12	0.13	0.35
10C13	0.18	1.25
10C14	0.2	1.53
10C15	0.18	1.86
20C11	0.54	3.28
20C12	0.49	0.47
20C13	0.49	1.29
20C14	0.52	1.3
20C15	0.44	0.48
30C11	0.94	0.62
30C12	1.08	0.21
30C13	0.87	0.33
30C14	0.9	0.27
30C15	0.81	0.2
50C11	3.15	0.35
50C12	2.09	0.13
50C13	2.44	1.91
50C14	2.33	1.99
50C15	2.16	0.61

Tabela 8.59: HNH-Futuro classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.25	2.94
10D17	0.26	2.61
10D18	0.23	3.10
10D19	0.23	3.20
10D20	0.28	3.14
20D16	0.82	2.40
20D17	0.72	1.70
20D18	0.74	2.42
20D19	1.02	2.75
20D20	0.73	1.83
30D16	1.44	1.93
30D17	2.01	2.79
30D18	1.53	2.04
30D19	1.45	1.28
30D20	1.40	1.99
50D16	3.61	0
50D17	4.08	1.15
50D18	3.72	1.89
50D19	3.50	2.69
50D20	3.82	1.13

Tabela 8.60: HNH-Futuro classe D: *gaps* e tempos

de segundo maior do que o tempo médio obtido pela heurística *GHS*, que é a solução utilizada por provedores de RDC.

Instância	Tempo (s)	Gap(%)
10A01	0.05	1.47
10A02	0.05	3.53
10A03	0.06	2.82
10A04	0.04	2.51
10A05	0.03	1.29
20A01	0.09	2.87
20A02	0.17	0.2
20A03	0.16	1.58
20A04	0.13	2.31
20A05	0.13	1.19
30A01	0.26	0.64
30A02	0.24	2.73
30A03	0.26	1.78
30A04	0.25	2.81
30A05	0.38	1.67
50A01	0.73	0.55
50A02	0.6	2.92
50A03	0.69	1.7
50A04	0.67	6.95
50A05	0.78	1.77

Tabela 8.61: HNH-Média classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.16	4.52
10B07	0.15	4.18
10B08	0.14	5.16
10B09	0.12	4.12
10B10	0.14	6.53
20B06	0.4	5.99
20B07	0.45	3.7
20B08	0.45	4.44
20B09	0.44	5.06
20B10	0.41	3.9
30B06	1.16	4.6
30B07	0.82	4.56
30B08	0.85	4.51
30B09	0.82	3.71
30B10	0.8	4.02
50B06	2.34	3.75
50B07	2.08	4.6
50B08	2.11	5.61
50B09	2.18	5.72
50B10	2.04	4.97

Tabela 8.62: HNH-Média classe B: *gaps* e tempos

As tabelas 8.61, 8.62, 8.63 e 8.64 mostram os resultados obtidos pela versão de *HNH* que utiliza o estimador de médias (*HNH-Média*), exposto na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Esta versão apresenta resultados idênticos aos obtidos pela versão *HCAGAP-Média* em termos de qualidade de solução e resultados melhores em termos de tempo, assim como ocorre para as versões *HCAGAP-Futuro* e *HNH-Futuro*.

As tabelas 8.65, 8.66, 8.67 e 8.68 mostram os resultados obtidos pela versão de *HNH* que utiliza o estimador *AELB* (*HNH-AELB*), exposto na Seção 6.3, para as instâncias

Instância	Tempo (s)	Gap(%)
10C11	0.18	10.48
10C12	0.15	5.25
10C13	0.17	8.86
10C14	0.18	11.61
10C15	0.18	20.54
20C11	0.52	19.25
20C12	0.49	6.25
20C13	0.46	10.1
20C14	0.48	11.02
20C15	0.44	6.06
30C11	0.88	7.77
30C12	1.09	5.91
30C13	0.87	6.21
30C14	0.89	5.1
30C15	0.82	5.56
50C11	2.32	5.45
50C12	2.1	5.45
50C13	2.53	14.77
50C14	2.4	14.9
50C15	2.24	8.72

Tabela 8.63: HNH-Média classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.30	8.68
10D17	0.28	8.30
10D18	0.25	6.23
10D19	0.22	7.38
10D20	0.30	8.65
20D16	1.13	8.49
20D17	0.68	6.11
20D18	0.78	7.45
20D19	0.82	7.88
20D20	0.82	6.70
30D16	1.95	8.69
30D17	1.44	8.22
30D18	2.12	8.05
30D19	1.52	6.96
30D20	1.41	8.32
50D16	3.81	10.51
50D17	5.35	6.02
50D18	3.80	321.73
50D19	4.22	3.34
50D20	4.30	5.54

Tabela 8.64: HNH-Média classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.05	1.47
10A02	0.06	3.59
10A03	0.05	3.11
10A04	0.04	2.59
10A05	0.03	1.39
20A01	0.08	2.87
20A02	0.2	0.23
20A03	0.17	1.51
20A04	0.12	2.36
20A05	0.14	1.17
30A01	0.3	0.66
30A02	0.22	2.75
30A03	0.25	1.84
30A04	0.28	2.83
30A05	0.29	1.71
50A01	0.71	0.54
50A02	0.61	2.95
50A03	0.67	1.7
50A04	0.67	6.95
50A05	1.04	1.77

Tabela 8.65: HNH-AELB classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.14	4.52
10B07	0.16	4.18
10B08	0.12	5.16
10B09	0.14	4.12
10B10	0.15	6.53
20B06	0.41	5.99
20B07	0.57	3.7
20B08	0.38	4.44
20B09	0.44	5.06
20B10	0.41	3.9
30B06	0.82	4.6
30B07	0.78	4.56
30B08	0.83	4.51
30B09	0.84	3.71
30B10	0.84	4.02
50B06	2.29	3.75
50B07	2.08	4.6
50B08	2.89	5.61
50B09	2.22	5.72
50B10	2.05	4.97

Tabela 8.66: HNH-AELB classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.17	6.77
10C12	0.15	4.62
10C13	0.15	6.86
10C14	0.19	7.73
10C15	0.16	10.76
20C11	0.68	8.92
20C12	0.46	4.27
20C13	0.43	6.45
20C14	0.51	7.44
20C15	0.41	4.6
30C11	0.9	5.61
30C12	0.8	5.08
30C13	1.02	4.93
30C14	0.9	4.23
30C15	1.06	4.55
50C11	2.31	4.39
50C12	2.1	4.85
50C13	3.02	8.13
50C14	2.36	8.33
50C15	2.16	6.2

Tabela 8.67: HNH-AELB classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.25	4.99
10D17	0.24	4.39
10D18	0.24	4.31
10D19	0.20	3.96
10D20	0.29	5.10
20D16	0.82	4.17
20D17	0.68	2.83
20D18	0.72	3.98
20D19	0.83	3.99
20D20	0.75	3.51
30D16	1.34	3.40
30D17	1.39	3.91
30D18	1.44	3.52
30D19	1.42	2.69
30D20	1.38	3.92
50D16	3.74	0.26
50D17	4.54	2.71
50D18	3.66	3.28
50D19	4.26	2.92
50D20	3.82	2.11

Tabela 8.68: HNH-AELB classe D: *gaps* e tempos

das classes A, B, C e D respectivamente. Esta versão, assim como esperado, apresenta resultados idênticos aos obtidos pela versão *HCAGAP-AELB* em termos de qualidade de solução e resultados melhores em termos de tempo, assim como ocorre para as versões anteriores de *HNH*.

Instância	Tempo (s)	Gap(%)
10A01	0.06	1.47
10A02	0.04	3.53
10A03	0.04	2.82
10A04	0.05	2.4
10A05	0.04	1.29
20A01	0.08	2.87
20A02	0.17	0.19
20A03	0.17	1.39
20A04	0.13	2.32
20A05	0.13	1.09
30A01	0.3	0.64
30A02	0.2	2.73
30A03	0.25	1.78
30A04	0.27	2.8
30A05	0.31	1.65
50A01	0.72	0.49
50A02	0.62	2.92
50A03	0.69	1.71
50A04	0.67	6.89
50A05	0.77	1.77

Tabela 8.69: HNH-AEBH classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.14	4.52
10B07	0.15	4.18
10B08	0.12	5.16
10B09	0.14	4.12
10B10	0.15	6.53
20B06	0.44	5.99
20B07	0.44	3.7
20B08	0.42	4.44
20B09	0.44	5.06
20B10	0.38	3.9
30B06	0.86	4.6
30B07	0.78	4.56
30B08	0.84	4.51
30B09	0.84	3.71
30B10	1.04	4.02
50B06	2.27	3.75
50B07	2.12	4.6
50B08	2.12	5.61
50B09	2.14	5.72
50B10	2.06	4.97

Tabela 8.70: HNH-AEBH classe B: *gaps* e tempos

As tabelas 8.69, 8.70, 8.71 e 8.72 mostram os resultados obtidos pela versão de *HNH* que utiliza o estimador *AEBH* (*HNH-AEBH*), também exposto na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Esta versão, assim como as demais versões de *HNH*, apresenta resultados idênticos aos obtidos pela versão *HCAGAP-AEBH* em termos de qualidade de solução e resultados melhores em termos de tempo.

As tabelas 8.73, 8.74, 8.75 e 8.76 mostram os resultados obtidos pela versão de *HNH*

Instância	Tempo (s)	Gap(%)
10C11	0.16	6.12
10C12	0.14	4.24
10C13	0.18	6.15
10C14	0.18	6.86
10C15	0.17	8.83
20C11	0.47	9.28
20C12	0.48	4.25
20C13	0.48	6.38
20C14	0.5	7.25
20C15	0.44	4.66
30C11	0.97	5.39
30C12	0.8	4.89
30C13	0.86	5
30C14	0.82	4.12
30C15	1.06	4.36
50C11	2.34	4.24
50C12	2.12	4.74
50C13	2.44	8.32
50C14	2.29	7.95
50C15	2.26	6.08

Tabela 8.71: HNH-AEBH classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.25	3.86
10D17	0.26	4.75
10D18	0.25	4.29
10D19	0.23	3.67
10D20	0.26	5.06
20D16	0.74	3.73
20D17	0.63	2.66
20D18	0.69	3.79
20D19	0.78	3.92
20D20	0.74	3.20
30D16	1.36	2.79
30D17	1.41	3.84
30D18	1.40	3.06
30D19	1.49	2.45
30D20	1.40	3.07
50D16	3.98	0.11
50D17	3.30	2.65
50D18	3.67	2.83
50D19	4.79	2.93
50D20	4.75	2.23

Tabela 8.72: HNH-AEBH classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.04	1.47
10A02	0.04	3.53
10A03	0.04	2.82
10A04	0.04	2.37
10A05	0.03	1.29
20A01	0.08	2.87
20A02	0.18	0.19
20A03	0.17	1.39
20A04	0.14	2.31
20A05	0.14	1.07
30A01	0.29	0.64
30A02	0.22	2.73
30A03	0.25	1.78
30A04	0.26	2.8
30A05	0.34	1.65
50A01	0.7	0.49
50A02	0.62	2.9
50A03	0.67	1.7
50A04	0.71	6.88
50A05	0.8	1.77

Tabela 8.73: HNH-Bote classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.13	4.52
10B07	0.16	4.18
10B08	0.16	5.16
10B09	0.14	4.12
10B10	0.13	6.53
20B06	0.39	5.99
20B07	0.45	3.7
20B08	0.43	4.44
20B09	0.42	5.06
20B10	0.4	3.9
30B06	0.89	4.6
30B07	1.07	4.56
30B08	0.86	4.51
30B09	0.8	3.71
30B10	0.82	4.02
50B06	2.35	3.75
50B07	2.12	4.6
50B08	2.2	5.61
50B09	2.23	5.72
50B10	2.13	4.97

Tabela 8.74: HNH-Bote classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.18	6.21
10C12	0.14	4.26
10C13	0.17	6.68
10C14	0.18	7.01
10C15	0.16	9.41
20C11	0.51	9.32
20C12	0.46	4.32
20C13	0.46	6.84
20C14	0.51	7.17
20C15	0.44	4.6
30C11	0.96	5.32
30C12	0.84	4.92
30C13	0.89	5.23
30C14	0.89	4.07
30C15	0.85	4.33
50C11	2.48	4.33
50C12	2.19	4.8
50C13	2.43	8.35
50C14	2.33	8.2
50C15	2.28	6.19

Tabela 8.75: HNH-Bote classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.28	4.45
10D17	0.24	4.66
10D18	0.21	4.71
10D19	0.21	4.01
10D20	0.30	5.19
20D16	1.02	3.97
20D17	0.68	2.82
20D18	0.74	4.12
20D19	0.73	4.01
20D20	0.74	3.46
30D16	1.35	2.66
30D17	1.40	4.25
30D18	1.41	3.20
30D19	1.48	2.80
30D20	1.43	3.31
50D16	3.54	0.15
50D17	3.24	2.83
50D18	3.53	2.99
50D19	3.48	2.96
50D20	3.75	2.48

Tabela 8.76: HNH-Bote classe D: *gaps* e tempos

que utiliza o estimador *Bote* (*HNH-Bote*), também exposto na Seção 6.3, para as instâncias das classes A, B, C e D respectivamente. Esta versão, apresenta resultados idênticos aos obtidos pela versão *HCAGAP-Bote* em termos de qualidade de solução e resultados melhores em termos de tempo, assim como ocorrido para as demais versões de *HNH*.

Como pode ser observado, cada uma das versões da heurística *HNH* (*HNH-Futuro*, *HNH-Média*, *HNH-AELB*, *HNH-AEBH* e *HNH-Bote*) superou a respectiva versão da heurística *HCAGAP*. Isto pode ser afirmado pois as versões de *HNH* apresentam os mesmos resultados que as versões de *HCAGAP* em um tempo computacional melhor.

8.7 Resultados dos Diferentes Estimadores

Esta seção apresenta os resultados obtidos pelos estimadores *Média*, *AELB*, *AEBH* e *Bote* para as instâncias de até 50 servidores. Os resultados da comparação entre os estimadores citados são apresentados na Tabela 8.77, na qual a primeira coluna apresenta os conjuntos de instâncias tratados. Cada entrada nesta primeira coluna representa um conjunto de 5 instâncias denotadas por um número, que representa o número de servidores das instâncias, e por uma letra que representa a classe das instâncias. Assim, a entrada 10A representa as instâncias da classe A com 10 servidores. As colunas dois e três representam a média dos *gaps* e o respectivo desvio padrão obtidos pelo estimador *Médias* para cada grupo de instâncias. As demais colunas representam a média dos gaps e o desvio padrão para os outros estimadores. A penúltima linha mostra a média dos gaps de cada estimador para todo o conjunto de instâncias. A última linha (# Melhores) mostra a quantidade de conjuntos em que cada estimador obteve a melhor média dos gaps.

Pela Tabela 8.77 é possível observar que não há um estimador que supere todos os outros em todos os casos. Neste sentido, não é possível identificar, de acordo com os testes realizados, um único estimador que seja o melhor entre eles. Contudo, podem-se destacar dois estimadores: i) o estimador *AEBH* que obteve a menor média de *gaps* e o maior valor de "# Melhores" e ii) o estimador *Bote* que, apesar de sua simplicidade, obteve a segunda menor média de *gaps* e o segundo maior valor de "# Melhores". Deste modo, apenas estes dois estimadores, *AEBH* e *Bote* serão utilizados nos demais testes desta tese.

	Médias		AELB		AEBH		Bote	
Instância	Média	Desvio	Média	Desvio	Média	Desvio	Média	Desvio
10A	2.32	0.84	2.43	0.88	**2.30**	0.84	**2.30**	0.84
10B	**4.90**	0.89	**4.90**	0.89	**4.90**	0.89	**4.90**	0.89
10C	11.35	5.07	7.35	1.99	**6.44**	1.47	6.72	1.65
10D	7.85	0.93	4.55	0.43	**4.33**	0.52	4.6	0.38
20A	1.63	0.92	1.63	0.92	**1.57**	0.94	**1.57**	0.94
20B	**4.62**	0.83	**4.62**	0.83	**4.62**	0.83	**4.62**	0.83
20C	10.54	4.79	**6.34**	1.74	6.36	1.82	6.45	1.84
20D	7.33	0.84	3.7	0.48	**3.46**	0.47	3.68	0.48
30A	**1.92**	0.80	1.96	0.79	**1.92**	0.80	**1.92**	0.80
30B	**4.28**	0.35	**4.28**	0.35	**4.28**	0.35	**4.28**	0.35
30C	6.11	0.91	4.88	0.47	**4.75**	0.46	4.78	0.49
30D	8.05	0.58	3.49	0.45	**3.04**	0.46	3.25	0.56
50A	2.78	2.22	2.78	2.22	2.76	2.21	**2.75**	2.20
50B	**4.93**	0.72	**4.93**	0.72	**4.93**	0.72	**4.93**	0.72
50C	9.86	4.23	6.38	1.62	**6.27**	1.64	6.37	4.23
50D	69.43	126.17	2.26	1.07	**2.15**	1.05	2.28	1.08
Média	9.87	-	4.15	-	4	-	4.09	-
# Melhores	5		5		14		8	

Tabela 8.77: Comparação dos gaps dos Estimadores

8.8 Resultados das Heurísticas Construtivas para as Instâncias de Grande Porte

Esta Seção apresenta os resultados obtidos pelas melhores heurísticas construtivas, apresentadas até o momento, para as instâncias com 100 ou mais servidores. De acordo com os testes apresentados nas seções 8.2, 8.3, 8.4, 8.5 e 8.6 pode-se concluir que as melhores heurísticas construtivas, considerando *gaps* e tempos computacionais, são *HNH* e *HC* e portanto somente estas duas heurísticas serão usadas nos testes com as instâncias de grande porte. Como mencionado na Seção 8.7, apenas os estimadores *AEBH* e *Bote* serão usados como mecanismos para estimar a demanda futura nestes testes.

Instância	Tempo (s)	Gap(%)
100C01	23.05	4.76
100C02	28.23	5.23
100C03	23.13	0.53
100C04	21.1	4.11
100C05	27.53	4.83
100D16	26.46	0.05
100D17	33.66	0.04
100D18	27.11	0.06
100D19	25.92	0.03
100D20	27.97	5.62

Tabela 8.78: HC-AEBH: *Gaps* e tempos para Instâncias de 100 Servidores

Instância	Tempo (s)	Gap(%)
200C01	119.75	0.15
200C02	84.25	0.09
200C03	97.75	0.09
200C04	84.85	0.09
200C05	85.42	0.05

Tabela 8.79: HC-AEBH: *Gaps* e tempos para Instâncias de 200 Servidores

Instância	Tempo (s)	Gap(%)
300C01	248.41	0.08
300C02	212.45	0.03
300C03	210.1	0.02
300C04	286.9	0.03
300C05	237.17	0.02

Tabela 8.80: HC-AEBH: *Gaps* e tempos para Instâncias de 300 Servidores

Instância	Tempo (s)	Gap(%)
400C01	388.42	0.03
400C02	424.19	0.02
400C03	506.02	0.01
400C04	440.32	0.02
400C05	455.4	0.02

Tabela 8.81: HC-AEBH: *Gaps* e tempos para Instâncias de 400 Servidores

As tabelas 8.78, 8.79, 8.80 e 8.81 mostram os resultados obtidos pela heurística *HC-AEBH* para as instâncias de grande porte com 100, 200, 300 e 400 servidores respectivamente. A primeira coluna mostra as instâncias tratadas. A segunda coluna expõe o tempo computacional em segundos. A terceira coluna mostra o *gap* entre a melhor solução encontrada por um dos métodos exatos e a solução produzida pela heurística. Note que apesar de usar um estimador que apresenta erros os gaps continuam baixo. Isto ocorre principalmente devido ao fato de que mesmo nas soluções encontradas pelos métodos exatos existem *backlogs* de requisições o que faz com otimizações no posicionamento das réplicas e na distribuição das requisições, que normalmente trariam grande benefício para a função objetivo, acabem por ser amortizados devido à grande penalidade paga nos *backlogs*.

Instância	Tempo (s)	Gap(%)
100C01	23.03	4.90
100C02	21.94	5.25
100C03	23.13	0.56
100C04	21.06	4.12
100C05	22.16	4.81
100D16	34.31	0.05
100D17	26.33	0.05
100D18	27.13	0.07
100D19	25.99	0.03
100D20	27.89	5.62

Tabela 8.82: HC-Bote: *Gaps* e tempos para Instâncias de 100 Servidores

As tabelas 8.82, 8.83, 8.84 e 8.85 mostram os resultados obtidos pela heurística *HC-Bote* para as instâncias de grande porte com 100, 200, 300 e 400 servidores respectivamente. A primeira coluna mostra as instâncias tratadas. A segunda coluna expõe o tempo computacional em segundos. A terceira coluna mostra o *gap* entre a melhor solução encontrada por um dos métodos exatos e a solução produzida pela heurística. O baixo valor do gap para instâncias com mais de 200 servidores pode ser observado com o uso do estimador *Bote*, assim como no estimador *AEBH*, e pode ser explicado do mesmo

Instância	Tempo (s)	Gap(%)
200C01	94.44	0.15
200C02	107.25	0.09
200C03	117.01	0.09
200C04	106.65	0.09
200C05	85.28	0.05

Tabela 8.83: HC-Bote: *Gaps* e tempos para Instâncias de 200 Servidores

Instância	Tempo (s)	Gap(%)
300C01	199.82	0.08
300C02	212.11	0.03
300C03	209.34	0.02
300C04	287	0.03
300C05	250.77	0.02

Tabela 8.84: HC-Bote: *Gaps* e tempos para Instâncias de 300 Servidores

modo. Assim como observado para as instâncias de pequeno porte, o estimador *AEBH* apresenta resultados melhores que o estimador *Bote*.

As tabelas 8.86, 8.87, 8.88 e 8.89 mostram os resultados obtidos pela heurística *HNH-AEBH* para as instâncias de grande porte com 100, 200, 300 e 400 servidores respectivamente. A primeira coluna mostra as instâncias tratadas. A segunda coluna expõe o tempo computacional em segundos. A terceira coluna mostra o *gap* entre a melhor solução encontrada por um dos métodos exatos e a solução produzida pela heurística.

As tabelas 8.90, 8.91, 8.92 e 8.93 mostram os resultados obtidos pela heurística *HNH-Bote* para as instâncias de grande porte com 100, 200, 300 e 400 servidores respectivamente. A primeira coluna mostra as instâncias tratadas. A segunda coluna expõe o tempo computacional em segundos. A terceira coluna mostra o *gap* entre a melhor solução encontrada por um dos métodos exatos e a solução produzida pela heurística.

Pelos resultados apresentados nesta seção pode-se observar que, assim como para as instâncias de pequeno porte, a heurística *HNH* leva ligeira vantagem sobre a heurística *HC*. Para os diferentes estimadores, ao comparar-se *HNH-AEBH* com *HNH-Bote*, percebe-se que o estimador *AEBH* é melhor em 4 das 25 instâncias. O estimador *Bote* é melhor que o estimador *AEBH* também em 4 instâncias e os dois estimadores apresentam os mesmos resultados para as demais instâncias.

Instância	Tempo (s)	Gap(%)
400C01	479.51	0.03
400C02	498.01	0.02
400C03	506.74	0.01
400C04	501.71	0.02
400C05	521.38	0.02

Tabela 8.85: HC-Bote: *Gaps* e tempos para Instâncias de 400 Servidores

Instância	Tempo (s)	Gap(%)
100C01	14.39	4.7
100C02	10.41	5.11
100C03	12.05	0.51
100C04	9.62	4.11
100C05	10.55	4.71
100D16	18.05	0.04
100D17	17.07	0.04
100D18	18.76	0.05
100D19	18.52	0.02
100D20	16.48	5.62

Tabela 8.86: HNH-AEBH: *Gaps* e tempos para Instâncias de 100 Servidores

Instância	Tempo (s)	Gap(%)
200C01	117.83	0.14
200C02	111.10	0.09
200C03	118.18	0.09
200C04	90.44	0.08
200C05	90.10	0.05

Tabela 8.87: HNH-AEBH: *Gaps* e tempos para Instâncias de 200 Servidores

Instância	Tempo (s)	Gap(%)
300C01	229.81	0.08
300C02	187.96	0.03
300C03	164.16	0.02
300C04	241.31	0.03
300C05	236.42	0.02

Tabela 8.88: HNH-AEBH: *Gaps* e tempos para Instâncias de 300 Servidores

Instância	Tempo (s)	Gap(%)
400C01	389.32	0.03
400C02	438.08	0.01
400C03	468.58	0.01
400C04	428.90	0.02
400C05	473.61	0.02

Tabela 8.89: HNH-AEBH: *Gaps* e tempos para Instâncias de 400 Servidores

Instância	Tempo (s)	Gap(%)
100C01	10.96	4.86
100C02	10.24	5.1
100C03	11.70	0.53
100C04	9.83	4.1
100C05	10.51	4.7
100D16	18.24	0.04
100D17	17.30	0.04
100D18	19.66	0.05
100D19	16.79	0.02
100D20	19.88	4.68

Tabela 8.90: HNH-Bote: *Gaps* e tempos para Instâncias de 100 Servidores

Instância	Tempo (s)	Gap(%)
200C01	120.22	0.14
200C02	97.26	0.09
200C03	117.65	0.09
200C04	92.51	0.08
200C05	90.76	0.05

Tabela 8.91: HNH-Bote: *Gaps* e tempos para Instâncias de 200 Servidores

Instância	Tempo (s)	Gap(%)
300C01	224.30	0.08
300C02	183.11	0.03
300C03	161.60	0.02
300C04	218.19	0.03
300C05	248.87	0.02

Tabela 8.92: HNH-Bote: *Gaps* e tempos para Instâncias de 300 Servidores

Instância	Tempo (s)	Gap(%)
400C01	469.94	0.03
400C02	428.16	0.02
400C03	390.43	0.01
400C04	458.96	0.02
400C05	460.41	0.02

Tabela 8.93: HNH-Bote: *Gaps* e tempos para Instâncias de 400 Servidores

8.9 Resultados das Heurísticas baseadas em Caminhos Mínimos

Esta seção apresenta os resultados para as heurísticas baseadas em caminhos mínimos apresentadas na Seção 6.5.5. Diferentemente das outras seções, nesta seção os resultados para as heurísticas são apresentados de maneira intercalada para facilitar a comparação entre os resultados.

Instância	Tempo (s)	Gap(%)
10A01	0	1.47
10A02	0.01	3.53
10A03	0.01	2.82
10A04	0	2.4
10A05	0.01	1.29
20A01	0.01	2.87
20A02	0.02	0.19
20A03	0.02	1.39
20A04	0.02	2.32
20A05	0.02	1.09
30A01	0.02	0.64
30A02	0.02	2.73
30A03	0.03	1.78
30A04	0.03	2.8
30A05	0.04	1.65
50A01	0.05	0.49
50A02	0.06	2.92
50A03	0.06	1.71
50A04	0.05	11.82
50A05	0.06	1.77

Tabela 8.94: SPA-AEBH para classe A: *gaps* e tempos

As tabelas 8.94, 8.95 e **??** mostram os resultados obtidos pelas heurísticas *SPA*, *ASPA* e $ALGA$, respectivamente, para as instâncias da classe A usando o estimador *AEBH*. Note que, para esta classe, as três heurísticas apresentaram o mesmo resultado em termos de qualidade de solução para todas as instâncias exceto uma ($50A04$) onde a heurística *SPA* leva desvantagem em relação às outras. Em termos de tempo computacional *SPA* apresenta os melhores resultados, seguida de $ALGA$ e *ASPA* respectivamente.

Instância	Tempo (s)	Gap(%)
10A01	0.03	1.47
10A02	0.01	3.53
10A03	0.02	2.82
10A04	0.02	2.4
10A05	0.01	1.29
20A01	0.02	2.87
20A02	0.09	0.19
20A03	0.07	1.39
20A04	0.08	2.32
20A05	0.06	1.09
30A01	0.12	0.64
30A02	0.08	2.73
30A03	0.1	1.78
30A04	0.1	2.8
30A05	0.12	1.65
50A01	0.29	0.49
50A02	0.24	2.92
50A03	0.27	1.71
50A04	0.26	7.4
50A05	0.32	1.77

Tabela 8.95: ASPA-AEBH para classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.02	1.47
10A02	0.01	3.53
10A03	0.02	2.82
10A04	0.02	2.4
10A05	0.01	1.29
20A01	0.03	2.87
20A02	0.08	0.19
20A03	0.07	1.39
20A04	0.06	2.32
20A05	0.06	1.09
30A01	0.1	0.64
30A02	0.08	2.73
30A03	0.12	1.78
30A04	0.07	2.8
30A05	0.14	1.65
50A01	0.25	0.49
50A02	0.21	2.92
50A03	0.29	1.71
50A04	0.3	7.4
50A05	0.31	1.77

Tabela 8.96: ALGA-AEBH para classe A: *gaps* e tempos

Ao comparar estas heurísticas com a heurística *HNH-AEBH* para a mesma classe, observa-se que elas tem o mesmo resultado em termos de qualidade de solução exceto para a instância $50A04$ onde *HNH-AEBH* apresenta o melhor resultado, seguido por *ASPA-AEBH* e *ALGA-AEBH* empatadas, e por último *SPA-AEBH*.

Instância	Tempo (s)	Gap(%)
10B06	0.03	4.52
10B07	0.03	4.18
10B08	0.03	5.16
10B09	0.04	4.12
10B10	0.04	6.53
20B06	0.08	5.99
20B07	0.08	3.7
20B08	0.06	4.44
20B09	0.07	5.06
20B10	0.07	3.9
30B06	0.12	4.6
30B07	0.12	4.56
30B08	0.1	4.51
30B09	0.11	3.71
30B10	0.09	4.02
50B06	0.22	3.75
50B07	0.2	4.6
50B08	0.21	5.61
50B09	0.26	5.72
50B10	0.24	4.97

Tabela 8.97: SPA-AEBH para classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.05	4.52
10B07	0.08	4.18
10B08	0.08	5.16
10B09	0.08	4.12
10B10	0.07	6.53
20B06	0.18	5.99
20B07	0.23	3.7
20B08	0.18	4.44
20B09	0.18	5.06
20B10	0.18	3.9
30B06	0.33	4.6
30B07	0.32	4.56
30B08	0.35	4.51
30B09	0.33	3.71
30B10	0.32	4.02
50B06	0.93	3.75
50B07	0.8	4.6
50B08	0.84	5.61
50B09	0.89	5.72
50B10	0.86	4.97

Tabela 8.98: ASPA-AEBH para classe B: *gaps* e tempos

As tabelas 8.97, 8.98 8.99 mostram os resultados obtidos pelas heurísticas *SPA*, *ASPA* e *ALGA* para as instâncias da classe B, ambas usando o estimador *AEBH*. Como ocorre nos demais casos, estas heurísticas não apresentam diferença em termos de qualidade de solução para esta classe de instâncias. Na que diz respeito ao tempo computacional, a heurística *SPA* novamente leva vantagem sobre as demais. A heurística *ALGA* ocupa a segunda posição em termos de tempo computacional e a heurística *ASPA* obtém os piores

Instância	Tempo (s)	Gap(%)
10B06	0.06	4.52
10B07	0.06	4.18
10B08	0.04	5.16
10B09	0.05	4.12
10B10	0.06	6.53
20B06	0.17	5.99
20B07	0.18	3.7
20B08	0.16	4.44
20B09	0.18	5.06
20B10	0.14	3.9
30B06	0.33	4.6
30B07	0.31	4.56
30B08	0.33	4.51
30B09	0.34	3.71
30B10	0.31	4.02
50B06	0.85	3.75
50B07	0.78	4.6
50B08	0.77	5.61
50B09	0.79	5.72
50B10	0.96	4.97

Tabela 8.99: ALGA-AEBH para classe B: *gaps* e tempos

resultados.

Instância	Tempo (s)	Gap(%)
10C11	0.06	6.12
10C12	0.05	4.24
10C13	0.06	6.15
10C14	0.08	6.86
10C15	0.06	8.83
20C11	0.16	9.28
20C12	0.1	4.25
20C13	0.14	6.38
20C14	0.16	7.25
20C15	0.06	4.66
30C11	0.16	5.39
30C12	0.13	4.89
30C13	0.12	5
30C14	0.14	4.12
30C15	0.11	4.36
50C11	0.26	4.24
50C12	0.2	4.74
50C13	0.47	8.32
50C14	0.38	7.95
50C15	0.3	6.08

Tabela 8.100: SPA-AEBH para classe C: *gaps* e tempos

As tabelas 8.100, 8.101 e 8.102 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* para as instâncias da classe C, todas usando o estimador *AEBH*. Note que não há diferença na qualidade das soluções produzidas por SPA e ASPA. A heurística ALGA obtém os piores gaps para esta classe de instâncias. Este resultado reforça que a maior dificuldade na resolução das instâncias da classe C está no PPR e não no PDR já que as heurísticas simplificada e otimizada para o PDR obtém o mesmo resultado.

Em termos de tempo computacional, ainda devido à simplicidade da heurística usada na resolução do PDR, a heurística *SPA* obtém vantagem, produzindo assim os melhores resultados seguida de *ALGA* e *ASPA* respectivamente.

Instância	Tempo (s)	Gap(%)
10C11	0.11	6.12
10C12	0.07	4.24
10C13	0.09	6.15
10C14	0.13	6.86
10C15	0.12	8.83
20C11	0.28	9.28
20C12	0.23	4.25
20C13	0.24	6.38
20C14	0.26	7.25
20C15	0.21	4.66
30C11	0.41	5.39
30C12	0.31	4.89
30C13	0.52	5
30C14	0.38	4.12
30C15	0.33	4.36
50C11	0.9	4.24
50C12	0.84	4.74
50C13	1.14	8.32
50C14	1.02	7.95
50C15	0.96	6.08

Tabela 8.101: ASPA-AEBH para classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.04	6.23
10C12	0.06	4.24
10C13	0.06	6.57
10C14	0.06	7.58
10C15	0.06	9.79
20C11	0.16	9.47
20C12	0.17	4.43
20C13	0.16	6.66
20C14	0.16	7.61
20C15	0.14	4.73
30C11	0.30	5.71
30C12	0.29	4.98
30C13	0.31	5.14
30C14	0.31	4.15
30C15	0.28	4.43
50C11	1.13	4.29
50C12	0.76	4.81
50C13	0.68	8.6
50C14	0.73	8.71
50C15	0.75	6.12

Tabela 8.102: ALGA-AEBH para classe C: *gaps* e tempos

Ao comparar os resultados com os obtidos pela heurística *HNH-AEBH* com os das heurísticas de caminhos mínimos, constata-se que as *HNH*, *SPA* e *ASPA* produzem soluções de mesma qualidade para a classe C de instâncias e que *ALGA* não é capaz de atingir os resultados das demais heurísticas, ficando com um *gap* médio de aproximadamente 6% do ótimo.

Instância	Tempo (s)	Gap(%)
10D16	0.1	1173.5
10D17	0.12	1201.35
10D18	0.11	4.5
10D19	0.11	1315.82
10D20	0.12	877.81
20D16	0.31	496.59
20D17	0.22	1709.09
20D18	0.25	382.21
20D19	0.27	5034.98
20D20	0.25	137.54
30D16	0.39	3.04
30D17	0.53	125.75
30D18	0.46	3.16
30D19	0.46	2.63
30D20	0.46	2783.48
50D16	0.98	576.74
50D17	0.84	2.85
50D18	0.8	1193.55
50D19	0.71	16.86
50D20	0.87	769.71

Tabela 8.103: SPA-AEBH para classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.14	3.87
10D17	0.16	4.76
10D18	0.14	4.33
10D19	0.14	3.69
10D20	0.21	5.13
20D16	0.6	91.02
20D17	0.32	13.36
20D18	0.37	3.9
20D19	0.41	4.1
20D20	0.55	3.28
30D16	0.68	2.87
30D17	1.02	3.94
30D18	0.77	3.07
30D19	0.73	2.52
30D20	0.63	115.39
50D16	2.27	98.68
50D17	1.73	2.69
50D18	1.69	2.99
50D19	1.7	2.95
50D20	1.8	274.59

Tabela 8.104: ASPA-AEBH para classe D: *gaps* e tempos

As tabelas 8.103, 8.104 e 8.105 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* para as instâncias da classe D, todas usando o estimador *AEBH*. Ao comparar-se *SPA* e *ASPA* percebe-se que a heurística *ASP* para resolução do PDR leva enorme vantagem sobre a heurística usada por *SPA*, uma vez que para todas as instâncias desta classe a heurística *ASPA* obteve melhores resultados em termos de qualidade de solução.

Instância	Tempo (s)	Gap(%)
10D16	0.05	4.5
10D17	0.08	4.76
10D18	0.06	4.34
10D19	0.05	3.69
10D20	0.05	5.15
20D16	0.18	93.83
20D17	0.2	346.46
20D18	0.18	4.18
20D19	0.18	461.36
20D20	0.21	3.64
30D16	0.36	2.94
30D17	0.28	4.15
30D18	0.34	3.1
30D19	0.35	2.66
30D20	0.33	357.08
50D16	0.82	45.48
50D17	0.86	2.91
50D18	0.93	3.09
50D19	1.29	2.96
50D20	1.24	250.33

Tabela 8.105: ALGA-AEBH para classe D: *gaps* e tempos

Ao comparar-se as heurística *ASPA* e *ALGA* percebe-se que *ALGA* leva uma pequena vantagem para a instância 10D19 e uma vantagem considerável para as instâncias 50D16 e 50D20. A heurística *ASPA* leva vantagem na outras 17 instâncias desta classe.

Em termos de tempo computacional, assim como ocorrido para o estimador *Conhecimento Futuro*, a heurística *ALGA* obtém os melhores resultados para esta classe, seguida da heurística *SPA* e da heurística *ASPA*.

Instância	Tempo (s)	Gap(%)
10A01	0.01	1.47
10A02	0.01	3.53
10A03	0	2.82
10A04	0.01	2.37
10A05	0.01	1.29
20A01	0	2.87
20A02	0.02	0.19
20A03	0.02	1.39
20A04	0.02	2.31
20A05	0.01	1.07
30A01	0.02	0.64
30A02	0.02	2.73
30A03	0.03	1.78
30A04	0.04	2.8
30A05	0.03	1.65
50A01	0.04	0.49
50A02	0.05	2.9
50A03	0.05	1.7
50A04	0.06	11.82
50A05	0.08	1.77

Tabela 8.106: SPA-Bote para classe A: *gaps* e tempos

As tabelas 8.106, 8.107 e 8.108 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA*, usando o estimador *Bote*, para as instâncias da classe A. Note que para para estas instâncias, a única instância na qual as heurística apresentam resultados diferentes é a instância 50A04, onde *ALGA* e *ASPA* empatam com o melhor resultado. Em termos

Instância	Tempo (s)	Gap(%)
10A01	0.02	1.47
10A02	0.01	3.53
10A03	0.01	2.82
10A04	0.02	2.37
10A05	0.01	1.29
20A01	0.02	2.87
20A02	0.06	0.19
20A03	0.08	1.39
20A04	0.07	2.31
20A05	0.06	1.07
30A01	0.1	0.64
30A02	0.09	2.73
30A03	0.1	1.78
30A04	0.09	2.8
30A05	0.13	1.65
50A01	0.27	0.49
50A02	0.25	2.9
50A03	0.26	1.7
50A04	0.28	7.4
50A05	0.32	1.77

Tabela 8.107: ASPA-Bote para classe A: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10A01	0.01	1.47
10A02	0.01	3.53
10A03	0.01	2.82
10A04	0.02	2.37
10A05	0.01	1.29
20A01	0.03	2.87
20A02	0.08	0.19
20A03	0.07	1.39
20A04	0.04	2.31
20A05	0.05	1.07
30A01	0.12	0.64
30A02	0.08	2.73
30A03	0.09	1.78
30A04	0.08	2.8
30A05	0.1	1.65
50A01	0.32	0.49
50A02	0.24	2.9
50A03	0.25	1.7
50A04	0.26	7.4
50A05	0.3	1.77

Tabela 8.108: ALGA-Bote para classe A: *gaps* e tempos

de tempo computacional, para esta classe de instâncias a heurística *SPA-Bote* obtém os melhores resultados, seguida por *ALGA-Bote* e *ASPA-Bote*.

Instância	Tempo (s)	Gap(%)
10B06	0.04	4.52
10B07	0.04	4.18
10B08	0.03	5.16
10B09	0.03	4.12
10B10	0.02	6.53
20B06	0.06	5.99
20B07	0.05	3.7
20B08	0.08	4.44
20B09	0.08	5.06
20B10	0.04	3.9
30B06	0.11	4.6
30B07	0.08	4.56
30B08	0.1	4.51
30B09	0.07	3.71
30B10	0.1	4.02
50B06	0.2	3.75
50B07	0.22	4.6
50B08	0.2	5.61
50B09	0.22	5.72
50B10	0.2	4.97

Tabela 8.109: SPA-Bote para classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10B06	0.06	4.52
10B07	0.06	4.18
10B08	0.07	5.16
10B09	0.05	4.12
10B10	0.07	6.53
20B06	0.18	5.99
20B07	0.22	3.7
20B08	0.18	4.44
20B09	0.2	5.06
20B10	0.18	3.9
30B06	0.32	4.6
30B07	0.31	4.56
30B08	0.36	4.51
30B09	0.34	3.71
30B10	0.32	4.02
50B06	0.88	3.75
50B07	0.82	4.6
50B08	0.82	5.61
50B09	0.83	5.72
50B10	0.8	4.97

Tabela 8.110: ASPA-Bote para classe B: *gaps* e tempos

As tabelas 8.109, 8.110 e 8.111 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA*, usando o estimador *Bote*, para as instâncias da classe B. Ao comparar os resultados destas heurísticas juntamente com os da heurística *HNH-Bote*, a heurística *SPA-Bote* leva vantagem por produzir soluções tão boas quanto as demais abordagens em um tempo computacional menor para a classe B de instâncias.

Para as instâncias da classe C, cujos resultados das heurísticas *SPA*, *ASPA* e *ALGA* encontram-se expostos nas tabelas 8.112, 8.113 e 8.113 respectivamente, a heurística *SPA-Bote* apresenta os melhores resultados em termos de tempo computacional. Em termos

Instância	Tempo (s)	Gap(%)
10B06	0.05	4.52
10B07	0.06	4.18
10B08	0.05	5.16
10B09	0.05	4.12
10B10	0.05	6.53
20B06	0.18	5.99
20B07	0.16	3.7
20B08	0.14	4.44
20B09	0.19	5.06
20B10	0.13	3.9
30B06	0.28	4.6
30B07	0.26	4.56
30B08	0.32	4.51
30B09	0.32	3.71
30B10	0.32	4.02
50B06	0.86	3.75
50B07	1	4.6
50B08	0.75	5.61
50B09	0.77	5.72
50B10	0.72	4.97

Tabela 8.111: ALGA-Bote para classe B: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.06	6.21
10C12	0.04	4.26
10C13	0.05	6.68
10C14	0.06	7.01
10C15	0.06	9.41
20C11	0.17	9.32
20C12	0.09	4.32
20C13	0.11	6.84
20C14	0.17	7.17
20C15	0.08	4.6
30C11	0.12	5.32
30C12	0.1	4.92
30C13	0.1	5.23
30C14	0.13	4.07
30C15	0.1	4.33
50C11	0.23	4.33
50C12	0.19	4.8
50C13	0.54	8.35
50C14	0.34	8.2
50C15	0.31	6.19

Tabela 8.112: SPA-Bote para classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.1	6.21
10C12	0.07	4.26
10C13	0.12	6.68
10C14	0.1	7.01
10C15	0.11	9.41
20C11	0.26	9.32
20C12	0.23	4.32
20C13	0.25	6.84
20C14	0.3	7.17
20C15	0.2	4.6
30C11	0.46	5.32
30C12	0.38	4.92
30C13	0.35	5.23
30C14	0.35	4.07
30C15	0.31	4.33
50C11	1.01	4.33
50C12	0.79	4.8
50C13	1.54	8.35
50C14	0.9	8.2
50C15	1.37	6.19

Tabela 8.113: ASPA-Bote para classe C: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10C11	0.06	6.27
10C12	0.05	4.26
10C13	0.05	7.4
10C14	0.05	7.63
10C15	0.05	10.86
20C11	0.16	9.67
20C12	0.16	4.5
20C13	0.16	6.97
20C14	0.17	7.47
20C15	0.16	4.82
30C11	0.36	5.5
30C12	0.34	4.94
30C13	0.34	5.21
30C14	0.3	4.15
30C15	0.3	4.38
50C11	0.85	4.37
50C12	0.74	4.78
50C13	0.7	8.69
50C14	0.86	9.12
50C15	0.74	6.22

Tabela 8.114: ALGA-Bote para classe C: *gaps* e tempos

de qualidade de solução as heurísticas *ASPA* e *SPA* encontram os melhores resultados para 18 das 20 instâncias e a heurística *ALGA* para as outras duas.

Instância	Tempo (s)	Gap(%)
10D16	0.16	1691.29
10D17	0.11	252.2
10D18	0.11	627.12
10D19	0.1	604.29
10D20	0.14	1776.45
20D16	0.31	1518.27
20D17	0.18	2132.85
20D18	0.26	710.37
20D19	0.25	1629.58
20D20	0.26	3.66
30D16	0.39	2.92
30D17	0.47	1520.12
30D18	0.45	3.31
30D19	0.4	2.98
30D20	0.43	1642.02
50D16	0.86	56.76
50D17	0.8	2.99
50D18	0.72	1237.98
50D19	0.74	16.88
50D20	0.85	2.88

Tabela 8.115: SPA-Bote para classe D: *gaps* e tempos

Instância	Tempo (s)	Gap(%)
10D16	0.16	4.47
10D17	0.17	4.69
10D18	0.15	4.82
10D19	0.13	4.05
10D20	0.19	5.24
20D16	0.39	12.31
20D17	0.44	2.89
20D18	0.42	4.2
20D19	0.4	41.01
20D20	0.51	3.56
30D16	0.73	2.75
30D17	0.74	4.37
30D18	0.74	3.22
30D19	0.93	2.87
30D20	0.7	378.62
50D16	1.62	53.75
50D17	1.59	2.85
50D18	1.67	3.15
50D19	1.69	2.98
50D20	1.76	2.64

Tabela 8.116: ASPA-Bote para classe D: *gaps* e tempos

As tabelas 8.115, 8.116 e 8.117 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA*, usando o estimador *Bote*, para as instâncias da classe D. Ao comparar os resultados destas heurísticas percebe-se que os melhores resultados em termos de qualidade de solução são obtidos por *ASPA* que obtém as melhores soluções para 14 das 20 instâncias desta classe. A heurística *ALGA* fica em segundo lugar com 6 melhores soluções. A heurística SPA não obteve nenhuma das melhores soluções para esta classe.

Ao comparar-se estes resultados resultados com os obtidos por *HNH-Bote*, percebe-se que *HNH* obtém os melhores resultados para 17 das 20 instâncias e que *ALGA* obtém

Instância	Tempo (s)	Gap(%)
10D16	0.06	5.02
10D17	0.07	4.56
10D18	0.06	4.81
10D19	0.04	3.92
10D20	0.06	5.56
20D16	0.19	179.71
20D17	0.18	2.95
20D18	0.17	4.11
20D19	0.17	98.94
20D20	0.19	3.72
30D16	0.4	3.14
30D17	0.3	4.55
30D18	0.4	3.36
30D19	0.41	3.06
30D20	0.34	266.37
50D16	0.8	43.69
50D17	0.83	3.09
50D18	0.96	3.23
50D19	0.96	3.01
50D20	0.92	2.95

Tabela 8.117: ALGA-Bote para classe D: *gaps* e tempos

os outros 3 melhores resultados. Em termos de tempo computacional a classificação das heurísticas fica a seguinte: com menor tempo médio está *ALGA* seguida de *SPA*, *ASPA* e *HNH*.

Os resultados desta seção mostram dois fatos: i) apesar de a estratégia usada pela heurística *SPA* para a resolução do PDR ser muito simples pode se comprovar que esta estratégia é muito eficiente tanto em termos de qualidade de solução como em termos de tempo computacional para diversos casos. Como esta estratégia é usada pela literatura [8, 13, 25] é de se esperar que ela possua bom desempenho em certos casos, especialmente onde há abundância de banda nos servidores. ii) As instâncias das classes A, B e C quando particionadas em dois subproblemas resultam em um PDR de fácil resolução uma vez que estratégias que não fazem divisão da demanda entre múltiplos servidores obtêm resultados tão bons quanto as estratégias que fazem esta divisão.

8.10 Resultados das Heurísticas de Caminho Mínimo para as Instâncias de Grande Porte

As Tabelas 8.118, 8.119 e 8.120 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *AEBH*. Ao comparar os resultados pode-se observar que *ASPA* obtém os melhores resultados para todas as instâncias exceto para a instância 100D19 onde *ALGA* obtém o melhor resultado. A heurística *SPA* empata com *ALGA* em 4 instâncias da classe C, o que mostra que esta classe, mesmo para instâncias grandes, apresenta um PDR de fácil resolução.

Instância	Tempo(s)	Gap(%)
100C01	1.96	4.7
100C02	1.83	5.11
100C03	2.42	20.86
100C04	2.26	4.11
100C05	1.9	4.71
100D16	5.3	158.7
100D17	5.61	52.93
100D18	7.42	64.42
100D19	4.9	119.16
100D20	4.03	152.09

Tabela 8.118: SPA-AEBH: Instâncias com 100 servidores - *gaps* e tempos

Instância	Tempo(s)	Gap(%)
100C01	3.42	4.7
100C02	3.34	5.11
100C03	4.8	0.51
100C04	3.13	4.11
100C05	3.31	4.71
100D16	7.05	25.87
100D17	7.72	0.04
100D18	9.6	0.06
100D19	7.21	43.05
100D20	6.18	5.63

Tabela 8.119: ASPA-AEBH: Instâncias com 100 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
100C01	3.30	4.76
100C02	3.25	5.23
100C03	3.24	0.53
100C04	3.03	4.11
100C05	4.11	4.83
100D16	3.67	27.07
100D17	3.64	0.05
100D18	3.86	5.81
100D19	3.54	28.61
100D20	3.75	5.63

Tabela 8.120: ALGA-AEBH: Instâncias com 100 servidores - *gaps* e tempos

Em termos de tempo computacional, a heurística *ALGA* obtém os melhores resultados seguida de *SPA* e de *ASPA*. Ao comparar-se os resultados obtidos com os de *HNH-AEBH* percebe-se que as quatro heurísticas obtém o mesmo resultado para a instância 100C04 e que *HNH* leva vantagem nas demais instâncias.

Instância	Tempo(s)	Gap(%)
100C01	1.94	4.86
100C02	1.83	5.1
100C03	2.33	20.88
100C04	1.62	4.1
100C05	1.86	4.7
100D16	6.51	104.45
100D17	5.56	52.94
100D18	6.79	64.42
100D19	4.51	94.32
100D20	4	119.21

Tabela 8.121: SPA-Bote: Instâncias com 100 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
100C01	3.4	4.86
100C02	3.27	5.1
100C03	4.62	0.53
100C04	3.14	4.1
100C05	3.32	4.7
100D16	8.69	0.04
100D17	8.15	0.05
100D18	9.33	0.07
100D19	7.19	16.2
100D20	6.22	4.69

Tabela 8.122: ASPA-Bote: Instâncias com 100 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
100C01	3.25	4.9
100C02	3.26	5.25
100C03	3.22	0.56
100C04	3.84	4.12
100C05	3.15	4.81
100D16	3.64	0.05
100D17	3.67	0.06
100D18	5.32	0.08
100D19	3.49	27.79
100D20	3.70	5.63

Tabela 8.123: ALGA-Bote: Instâncias com 100 servidores - *gaps* e tempos

As Tabelas 8.121, 8.122 e 8.123 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *Bote*. Ao comparar-se os resultados claramente percebe-se que a heurística ASPA leva vantagem em termos de qualidade de solução uma vez que produz soluções melhores ou iguais às produzidas pelas demais abordagens para todas as instâncias. Ao comparar as heurísticas *SPA* e *ALGA*, percebe-se que *ALGA* leva vantagem para todas as instâncias da classe D e para a instância 100C03. Nas demais instâncias da classe C *SPA* leva vantagem sobre *ALGA*. Em termos de tempo computacional, os melhores resultados foram obtidos pela heurística *ALGA*, seguida de *SPA* e por último se encontra *ASPA*.

Ao comparar-se os resultados com os de *HNH-Bote*, percebe-se que *ASPA* obtém resultados iguais ou pouco piores para 9 instâncias e bem piores para a instância 100D19. Deste modo, *HNH* fornece melhores resultados mas, em termos de tempo computacional, é a heurística que fornece o pior resultado.

Tabela 8.124: spa-des2par

Instância	tempo	Gap(%)
200C01	8.84	13.72
200C02	6.58	15.39
200C03	7.34	12.11
200C04	6.56	13.22
200C05	6.62	13.07

Tabela 8.125: SPA-AEBH: Instâncias com 200 servidores - *gaps* e tempos

Tabela 8.126: aspades2p

Instância	tempo	Gap(%)
200C01	18.69	0.14
200C02	17.39	0.09
200C03	14.52	0.06
200C04	12.99	0.08
200C05	12.88	0.02

Tabela 8.127: ASPA-AEBH: Instâncias com 200 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
200C01	13.59	0.15
200C02	12.02	0.09
200C03	13.8	0.06
200C04	16.21	0.09
200C05	12.09	0.02

Tabela 8.128: ALGA-AEBH: Instâncias com 200 servidores - *gaps* e tempos

As Tabelas 8.125, 8.127 e 8.128 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *AEBH*. Ao comparar-se os resultados das três heurísticas percebe-se que a heurística *ASPA* leva vantagem sobre as demais pois produz soluções melhores ou iguais às produzidas pelas demais heurísticas. A heurística *ALGA* obtém os segundo lugar em termos de qualidade de solução, com resultados muito próximos aos de *ASPA*. A heurística *SPA* obteve o pior resultado em termos de qualidade de solução para estes testes. Em termos de tempo computacional, *SPA* obteve os melhores resultados, seguida por *ALGA* e *ASPA*.

É importante mencionar que, as heurísticas *ALGA* e *ASPA* obtiveram resultados melhores que *HNH-AEBH* para duas destas instâncias, a saber 200C03 e 200C05. Este resultado prova que o PPRDR não pode ser particionado em dois subproblemas (PPR e PDR) independentes. Isto ocorre porque uma vez que as estratégias de resolução do PDR são diferentes as soluções geradas também podem ser diferentes. Como uma solução do

PDR de um período t influencia diretamente na demanda por conteúdos do período $t+1$, pequenas diferenças entre as soluções do PDR de um período t acabam gerando pequenas diferenças na demanda por conteúdo do $t+1$, que por sua vez provocam pequenas diferenças na solução do PPR. Assim, as pequenas diferenças no atendimento das requisições entre a heurística *ASP* e o modelo de fluxo em rede para estas instância fazem com que tanto a formulação *AGAP* quanto a heurística *GAS*, nas heurísticas *ASPA* e *ALGA* respectivamente, optem por um posicionamento de réplicas diferente do encontrado por *HNH*, o que por uma feliz coincidência, acaba afetando positivamente a função objetivo para outros períodos. É importante mencionar que apesar de possível, este fato é pouco provável uma vez que, para as todas as instâncias testadas nesta tese, considerando todas as combinações de algoritmos testados, ele ocorre em menos de 1% dos casos. Testes adicionais foram feitos para verificar se este fato ocorre utilizando um mesmo posicionamento de réplicas para todos os períodos e constatou-se que dado um mesmo posicionamento de réplicas para as diferentes abordagens de resolução do PDR o modelo de fluxo em rede sempre apresenta soluções melhores ou iguais às demais abordagens, como esperado.

Instância	tempo	Gap(%)
200C01	8.54	13.72
200C02	6.51	15.39
200C03	7.31	12.11
200C04	6.59	13.22
200C05	6.37	13.07

Tabela 8.129: SPA-Bote: Instâncias com 200 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
200C01	17.14	0.14
200C02	12.74	0.09
200C03	14.65	0.06
200C04	16.63	0.08
200C05	12.89	0.02

Tabela 8.130: ASPA-Bote: Instâncias com 200 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
200C01	17.26	0.15
200C02	11.95	0.09
200C03	17.89	0.06
200C04	12.01	0.09
200C05	15.16	0.02

Tabela 8.131: ALGA-Bote: Instâncias com 200 servidores - *gaps* e tempos

As Tabelas 8.129, 8.130 e 8.131 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *Bote* para as instâncias de 200 servidores. Ao comparar-se os resultados das três heurísticas baseadas em caminho mínimo, considerando a qualidade das soluções, percebe-se que *ASPA* e *ALGA* têm desempenho idêntico para todas as instâncias exceto a instância 200C01, onde *ASPA* leva uma pequena vantagem. A heurística *SPA*

fica em média 13% atrás das demais. Note que para as instâncias 200C03 e 200C05, *ASPA* e *ALGA* conseguem atingir resultados melhores que a heurística *HNH-Bote*. Em termos de tempo computacional as heurísticas podem ser classificadas na seguinte ordem de tempo médio: em primeiro está a heurística *SPA*, em segundo a heurística *ALGA* e por último a heurística *ASPA*.

Instância	tempo	Gap(%)
300C01	16.68	10.89
300C02	15.71	9.4
300C03	19.28	8.01
300C04	17.28	9.63
300C05	19.71	8.6

Tabela 8.132: SPA-AEBH: Instâncias com 300 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
300C01	37.57	0.07
300C02	36.9	0.03
300C03	37.01	0.02
300C04	38.42	0.02
300C05	48.46	0.02

Tabela 8.133: ASPA-AEBH: Instâncias com 300 servidores - *gaps* e tempos

Instância	Tempo (s)	Gap(%)
300C01	32.09	0.08
300C02	36.03	0.03
300C03	36.68	0.02
300C04	37.44	0.02
300C05	39.75	0.02

Tabela 8.134: ALGA-AEBH: Instâncias com 300 servidores - *gaps* e tempos

As Tabelas 8.132, 8.133 e 8.134 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *AEBH* para as instâncias de 300 servidores. Ao comparar-se As heurísticas percebe-se que *ASPA* e *ALGA* obtém soluções de qualidade muito superior às de *SPA*. Em termos de tempo computacional, *SPA* leva grande vantagem sobre *ALGA* que possui a segunda melhor média de tempo. *ASPA* apresenta o pior resultado em termos de tempo computacional. Note que apesar de resolver o PPR de maneira heurística, *ALGA* não consegue bons resultados em termos de tempo quando comparada com SPA que usa uma formulação matemática para a resolução do PPR. Isto mostra que a maior parte do tempo computacional de *ALGA* e *ASPA* é gasta na resolução do PDR. Comparando *ALGA* e *ASPA* com *HNH-AEBH*, percebe-se que *ASPA* produz os melhores resultados para as instâncias 300C01 e 300C04. *ALGA* empata com *ASPA* para a instância 300C04. Para as demais instâncias as três heurísticas obtém os mesmos resultados.

As Tabelas 8.135, 8.136 e 8.138 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *Bote* para as instâncias de 300 servidores. Observando os

Instância	tempo	Gap(%)
300C01	16.13	10.89
300C02	15.57	9.40
300C03	14.00	8.01
300C04	17.16	9.63
300C05	21.76	8.60

Tabela 8.135: SPA-Bote: Instâncias com 300 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
300C01	38.70	0.07
300C02	36.60	0.03
300C03	36.90	0.02
300C04	38.43	0.02
300C05	48.60	0.02

Tabela 8.136: ASPA-Bote: Instâncias com 300 servidores - *gaps* e tempos

resultados apresentados pode-se chegar às mesmas conclusões obtidas para as instâncias de 300 servidores usando o estimador *AEBH*.

As Tabelas 8.140, 8.141 e 8.142 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *AEBH* para as instâncias de 400 servidores. Note que ao comparar as heurísticas de caminho mínimo, novamente *ASPA* e *ALGA* levam vantagem sobre *SPA* em termos de qualidade de solução. Em termos de tempo computacional SPA obtém os melhores resultados seguida por *ALGA* e por *ASPA* respectivamente. Ao comparar-se os resultados de *ASPA* e *ALGA* com os de *HNH-AEBH* para estas instâncias, observa-se que *ASPA* e *ALGA* conseguem resultados melhores para a instância 400C02, onde estas heurísticas conseguem um resultado muito próximo da melhor solução exata.

As Tabelas 8.143, 8.144 e 8.145 mostram os resultados obtidos por *SPA*, *ASPA* e *ALGA* usando o estimador *Bote* para as instâncias de 400 servidores. Novamente *ASPA* e *ALGA* obtém resultados melhores que os produzidos por *SPA* em termos de qualidade de solução. Em termos de tempo computacional *SPA* leva vantagem sobre as outras heurísticas. Comparando as soluções produzidas por *ALGA* e *ASPA* com as de *HNH-Bote* percebe-se que ALGA e ASPA obtém melhores soluções para as instâncias 400C02 e 400C04.

Os resultados apresentados nesta seção mostram que heurísticas baseadas em caminho

Tabela 8.137: algadelta

Instância	Tempo (s)	Gap(%)
300C01	32.09	0.08
300C02	47.75	0.03
300C03	36.41	0.02
300C04	37.56	0.03
300C05	39.69	0.02

Tabela 8.138: ALGA-Bote: Instâncias com 300 servidores - *gaps* e tempos

Tabela 8.139: spades2par

Instância	tempo	Gap(%)
400C01	31.20	9.07
400C02	32.20	6.60
400C03	27.76	7.22
400C04	28.90	7.60
400C05	42.42	9.71

Tabela 8.140: SPA-AEBH: Instâncias com 400 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
400C01	83.05	0.04
400C02	88.11	0.00
400C03	92.35	0.02
400C04	68.88	0.01
400C05	98.87	0.02

Tabela 8.141: ASPA-AEBH: Instâncias com 400 servidores - *gaps* e tempos

Instância	Tempo (s)	0
400C01	63.05	0.04
400C02	95.70	0.00
400C03	91.80	0.02
400C04	67.66	0.01
400C05	68.14	0.02

Tabela 8.142: ALGA-AEBH: Instâncias com 400 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
400C01	32.35	9.07
400C02	31.35	6.60
400C03	26.96	7.22
400C04	28.72	7.60
400C05	27.83	9.71

Tabela 8.143: SPA-Bote: Instâncias com 400 servidores - *gaps* e tempos

Instância	tempo	Gap(%)
400C01	76.89	0.04
400C02	81.42	0.00
400C03	69.72	0.02
400C04	68.92	0.01
400C05	89.76	0.02

Tabela 8.144: ASPA-Bote: Instâncias com 400 servidores - *gaps* e tempos

Instância	Tempo (s)	0
400C01	83.01	0.04
400C02	72.45	0.00
400C03	91.02	0.02
400C04	66.56	0.01
400C05	79.94	0.02

Tabela 8.145: ALGA-Bote: Instâncias com 400 servidores - *gaps* e tempos

mínimo são uma boa alternativa ao uso do modelo de fluxo em rede para instâncias onde os servidores possuem banda abundante, independentemente do tamanho das instâncias. Para casos onde boas soluções são desejadas, as heurísticas ALGA e ASPA são a melhor opção. Para casos onde se deseja um baixo tempo de execução a heurística SPA é a melhor opção. Já para as instâncias onde a banda dos servidores é restrita, o modelo de fluxo em rede ainda é a melhor opção para resolver o PDR visto que para algumas instâncias da classe D, os gaps produzidos pelas heurísticas de caminho mínimo são muito superiores aos produzidos pelo modelo de fluxo em rede.

8.11 Resultados das Meta-Heurísticas *ILS*

Esta seção apresenta os resultados para as duas versões da meta-heurística *ILS*. A Tabela 8.146 mostra o comparativo dos *gaps* apresentados pelos dois algoritmos para instâncias de com 10 servidores das classes C e D. A primeira coluna indica as instâncias abordadas. As colunas 2 e 3 mostram respectivamente o melhor gap e o gap médio obtidos pelo algoritmo *ILS*. Já as colunas 4 e 5 apresentam o melhor gap e o *gap* médio obtidos pela heurística *ILSAM*. Em negrito estão os melhores *gaps* encontrados por estes dois algoritmos. Note que o pior *gap* encontrado pelas meta-heurísticas é menor que 2.5%, o que indica que as heurísticas possuem um bom desempenho para estas instâncias. Também é importante notar que em apenas uma instância o algoritmo *ILSAM* não obteve o melhor resultado e que em diversos casos, marcados com "*", o resultado médio apresentado pelo algoritmo *ILSAM* possui qualidade superior ao melhor resultado encontrado pelo *ILS*. Outro ponto a ser destacado é que os resultados médios do *ILSAM* são superiores aos resultados médios obtidos pelo *ILS* em todos os casos.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
10C11	0.08	0.12	**0.07**	0.10
10C12*	0.07	0.09	**0.06**	0.07
10C13	**0.15**	0.23	0.18	0.22
10C14*	0.40	0.44	**0.28**	0.36
10C15	0.36	0.46	**0.26**	0.37
10D16*	2.27	2.29	**1.93**	2.09
10D17*	2.00	2.05	**1.72**	1.82
10D18*	1.96	1.98	**1.61**	1.69
10D19*	1.54	1.59	**1.29**	1.39
10D20*	1.75	1.88	**1.52**	1.65

Tabela 8.146: Gaps: ILS X ILSAM - 10 Servidores

A Tabela 8.147 mostra os resultados relacionados aos tempos computacionais para os dois algoritmos. Na coluna um estão as instâncias examinadas. As colunas dois e três expõem os resultados obtidos pelo algoritmo *ILS*, assim como as colunas quatro e cinco

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
10C11	5812.00	6357.33	3843.00	4907.00
10C12	2590.00	1400.67	1248.00	1447.67
10C13	7096.00	7027.67	4472.00	5605.67
10C14	6844.00	6705.67	7064.00	7085.00
10C15	6757.00	6844.33	7043.00	6924.67
10D16	6903.00	7055.67	6911.00	6820.00
10D17	6869.00	7052.67	7094.00	6966.00
10D18	7121.00	7077.00	7083.00	7124.00
10D19	7119.00	7113.67	7205.00	7130.33
10D20	7010.00	7096.67	6925.00	6965.33

Tabela 8.147: Tempos: ILS X ILSAM - 10 Servidores

apresentam os resultados do *ILSAM*. Na coluna dois está exposto o tempo computacional necessário para encontrar o melhore resultado do algoritmo *ILS*. Na coluna três está exposto a média dos tempos necessários pelo *ILS* para encontrar os melhores resultados. As colunas quatro e cinco possuem significado análogo às colunas três e quatro, no entanto, as colunas quatro e cinco apresentam os resultados para o algoritmo *ILSAM*. Note que os dois algoritmos se alternam nos resultados de tempo tanto para o melhor tempo quanto para as médias. Este fato é esperado uma vez que os dois algoritmos possuem comportamento semelhante.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
20C11*	2.65	2.69	**2.18**	2.27
20C12	**0.27**	0.31	0.29	0.3
20C13*	1.05	1.06	**0.97**	1.01
20C14	**1.10**	1.14	1.13	1.17
20C15	**0.34**	0.39	0.37	0.38
20D16*	1.96	2.03	**1.88**	1.89
20D17*	1.50	1.54	**1.44**	1.46
20D18*	2.18	2.2	**1.99**	2.02
20D19	2.47	2.48	**2.29**	2.82
20D20*	1.72	1.74	**1.64**	1.67

Tabela 8.148: Gaps: ILS X ILSAM - 20 Servidores

A Tabela 8.148 mostra os resultados para as instâncias de 20 servidores. Destaque deve ser dado para a instância 20C14, onde o algoritmo *ILS* obteve o melhor resultado e a melhor média. O *ILSAM* obteve a melhor média todas as outras instâncias exceto um (20D19) e apenas para outras duas (20C12 e 20C15) o *ILSAM* não obteve os melhores resultados. É importante notar que, assim como nas instâncias de 10 servidores, em diversas instâncias, também marcadas com "*", o algoritmo *ILSAM* obteve um resultado médio cuja a qualidade é superior ao do melhor resultado do *ILS*.

A Tabela 8.149 mostra os resultados obtidos pelas duas meta-heurísticas no que tange o tempo computacional para as instâncias de vinte servidores. Note que resultados semelhantes aos relatados para as instâncias de dez servidores também são observados aqui.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
20C11	7225.00	6995.67	7101.00	7091.33
20C12	7209.00	7018.00	5973.00	6422.67
20C13	7002.00	7098.00	6710.00	6786.67
20C14	5488.00	6453.00	7028.00	7078.33
20C15	6363.00	6508.33	7150.00	6957.67
20D16	7025.00	6987.00	7069.00	6823.33
20D17	6698.00	6662.33	6714.00	6491.00
20D18	7144.00	6870.67	6477.00	6847.67
20D19	7058.00	6922.67	7026.00	6557.00
20D20	5916.00	6533.00	6843.00	6573.33

Tabela 8.149: Tempos: ILS X ILSAM - 20 Servidores

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
30C11*	0.59	0.60	**0.55**	0.58
30C12*	0.19	0.20	**0.18**	0.19
30C13	**0.27**	0.28	0.28	0.29
30C14*	0.25	0.26	**0.23**	0.24
30C15*	0.18	0.18	**0.17**	0.18
30D16*	1.69	1.71	**1.5**	1.58
30D17*	2.58	2.61	**2.36**	2.41
30D18*	1.87	1.90	**1.78**	1.79
30D19*	1.20	1.23	**1.17**	1.19
30D20*	1.78	1.79	**1.61**	1.63

Tabela 8.150: Gaps: ILS X ILSAM - 30 Servidores

A Tabela 8.150 mostra os resultados obtidos pelas duas meta-heurísticas para as instâncias de trinta servidores. Destaque deve ser dado para a instância 30C13, onde o algoritmo *ILS* obteve o melhor resultado e a melhor média. Para todas as demais instâncias o algoritmo *ILSAM* obteve uma média de qualidade superior ao melhor resultado obtido pelo *ILS*.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
30C11	4358.00	5034.33	4943.00	5811.00
30C12	3341.00	5565.67	7103.00	4586.00
30C13	6417.00	6764.00	6029.00	6538.67
30C14	6103.00	6070.00	7005.00	5304.00
30C15	3991.00	3911.00	6276.00	6722.67
30D16	7154.00	7095.33	4995.00	6083.33
30D17	7091.00	6964.67	7266.00	6866.00
30D18	6731.00	6652.67	4973.00	5234.00
30D19	6165.00	5736.67	6762.00	6657.00
30D20	6312.00	6453.67	6750.00	6914.67

Tabela 8.151: Tempos: ILS X ILSAM - 30 Servidores

A Tabela 8.151 mostra os tempos computacionais para as instâncias de trinta servidores. O fato dos algoritmos se alternarem nos melhores resultados, tanto da média dos tempos como no tempo para encontrar os melhores resultados, também pode ser observado aqui.

A Tabela 8.152 mostra os resultados dos algoritmos *ILS* e *ILSAM* para as instâncias com cinquenta servidores. O algoritmo *ILSAM* conseguiu a melhor média em nove das

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
50C11*	0.342	0.344	**0.334**	0.341
50C12*	0.125	0.125	**0.124**	0.125
50C13	1.886	1.898	**1.881**	1.895
50C14*	1.978	1.981	**1.970**	1.977
50C15	**0.605**	0.609	0.609	0.611
50D16	-0.035	-0.025	**-0.043**	-0.032
50D17*	1.135	1.140	**1.119**	1.132
50D18*	1.806	1.819	**1.688**	1.734
50D19*	2.685	2.685	**2.679**	2.681
50D20*	1.081	1.095	**1.028**	1.051

Tabela 8.152: Gaps: ILS X ILSAM - 50 Servidores

dez instâncias. O *ILS* conseguiu apenas um melhor resultado para a instância 50C15, instância esta para qual o *ILS* também conseguiu a melhor média. Destaque deve ser dado para a instância 50D16 na qual os dois algoritmos foram capazes de encontrar soluções melhores que a encontrada pelo método exato. É importante notar também que para estas instâncias, o algoritmo *ILSAM* apresentou resultados médios com qualidade superior ao melhor resultado do algoritmo *ILS*.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
50C11	2488.00	2455.67	2792.00	1123.67
50C12	100.00	108.33	90.00	89.00
50C13	5666.00	4578.33	5038.00	4842.00
50C14	5429.00	4410.33	80.00	1365.33
50C15	5097.00	4439.67	3908.00	3393.67
50D16	6257.00	6502.33	5913.00	6028.00
50D17	4182.00	5014.00	7309.00	4249.67
50D18	5553.00	5049.67	6574.00	6683.67
50D19	2809.00	3365.33	3057.00	4299.67
50D20	6954.00	6683.00	6694.00	6772.33

Tabela 8.153: Tempos: ILS X ILSAM - 50 Servidores

A Tabela 8.153 mostra os tempos computacionais para as instâncias de cinquenta servidores. A alternância dos algoritmos nos melhores resultados, tanto da média dos tempos como no tempo para encontrar os melhores resultados, novamente pode ser observada.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
100C01	**-0.0011**	-0.0004	**-0.0011**	-0.0004
100C02	**0.0000**	0.0000	**0.0000**	0.0000
100C03	**-0.0004**	-0.0002	**-0.0004**	-0.0002
100C04*	0.0000	0.0000	**-0.0047**	-0.0016
100C05	**-0.0002**	-0.0001	**-0.0002**	-0.0001
100D16	-0.0019	-0.0014	**-0.0023**	-0.0018
100D17*	-0.00006	-0.00003	**-0.00022**	-0.00009
100D18*	-0.0003	-0.0001	**-0.0003**	-0.0003
100D19	-0.0020	-0.0009	**-0.0026**	-0.0017
100D20*	-0.00245	-0.00098	**-0.00377**	-0.00312

Tabela 8.154: Gaps: ILS X ILSAM - 100 Servidores

A Tabela 8.154 mostra os resultados dos algoritmos *ILS* e *ILSAM* para as instâncias com cem servidores. Pela tabela é possível perceber que os dois algoritmos conseguem

atingir os mesmos resultados para todas as instâncias da classe C exceto para a instância 100C04, na qual o o algoritmo *ILSAM* obtém melhor desempenho, sendo que a média dos resultados obtidos pelo *ILSAM* possui qualidade superior ao melhor valor encontrado pelo *ILS* para esta instância. Para a classe D O algoritmo *ILSAM* obtém as melhores soluções e as melhores médias em todas as instâncias. Também é importante notar que ambos os algoritmos conseguem, em alguns casos encontrar resultados melhores que o método exato para estas instâncias.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
100C01	1274.00	577.00	1273.00	601.00
100C02	190.00	281.33	249.00	302.67
100C03	520.00	285.00	395.00	260.67
100C04	144.00	174.67	6578.00	2356.33
100C05	3236.00	1292.33	3205.00	1295.67
100D16	4313.00	4609.33	3428.00	4544.00
100D17	6600.00	4257.33	6449.00	4054.00
100D18	2542.00	3375.33	5903.00	5001.00
100D19	6036.00	6072.67	1912.00	4016.67
100D20	5992.00	4487.33	7612.00	6355.33

Tabela 8.155: Tempos: ILS X ILSAM - 100 Servidores

A Tabela 8.155 mostra os tempos computacionais para as instâncias de cem servidores. A alternância dos algoritmos nos melhores resultados, tanto da média dos tempos como no tempo para encontrar os melhores resultados, novamente pode ser observada.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
200C01	**0.000000**	0.000000	**0.000000**	0.000000
200C02*	0.000000	0.000000	**-0.000008**	-0.000003
200C03*	0.000000	0.000000	**-0.000024**	-0.000009
200C04	**0.000000**	0.000000	**0.000000**	0.000000
200C05	**-1.630154**	-1.629840	-1.629931	-1.629559

Tabela 8.156: Gaps: ILS X ILSAM - 200 Servidores

A Tabela 8.156 mostra os resultados dos algoritmos *ILS* e *ILSAM* para as instâncias com duzentos servidores. Pela tabela é possível perceber que o algoritmo *ILSAM* leva vantagem em todas as instâncias exceto para a instância 200C05 onde o *ILS* obtém a melhor solução e a melhor média. Também é importante notar que ambos os algoritmos conseguem, em alguns casos encontrar resultados melhores que o método exato para estas instâncias.

A Tabela 8.157 mostra os tempos computacionais para as instâncias de duzentos servidores. A alternância dos algoritmos para encontrar os melhores resultados e as melhores médias, que pode ser observada nas instâncias até 100 servidores não ocorre para estas instâncias, sendo que o algoritmo *ILS* tende a ser mais rápido tanto na média dos tempos quanto no tempo para encontrar as melhores soluções.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
200C01	2267.00	3305.67	2283.00	3491.67
200C02	1597.00	1684.33	3900.00	2662.00
200C03	1718.00	2620.33	2486.00	2759.67
200C04	1792.00	2472.33	1506.00	2378.67
200C05	1336.00	1814.00	2236.00	2488.33

Tabela 8.157: Tempos: ILS X ILSAM - 200 Servidores

Um fato importante a ser mencionado é que o tempo médio de cada iteração não cresce linearmente com o tamanho da instância. Isto faz com que o tempo computacional de cada iteração para as instâncias com mais de 100 servidores aumente muito, o que pode fazer com que os algoritmos não tenham um comportamento compatível com o que foi observado para as instâncias menores. Para verificar se isto realmente ocorre, foram feitas para as instâncias de 100 e 200 servidores, testes com tempo máximo para início de uma iteração até 5 vezes maior que o reportado. Os resultados apresentados pelos algoritmos com 10 horas e com 2 horas não apresentam diferença, o que pode significar que os resultados obtidos para estas instâncias estão fortemente relacionados com a solução inicial fornecida.

Outro fator importante a ser mencionado é o fato de que a medida que o tamanho das instâncias aumenta, o número de iterações que podem ser feitas no limite de duas horas diminui como pode ser visto na Tabela 8.158 que mostra na primeira coluna o tamanho das instâncias, em número de servidores e na segunda coluna o número médio de iterações.

Tamanho	Iterações
10	1265
20	355
30	143
50	51
100	21
200	2
300	1
400	1

Tabela 8.158: Número médio de iterações por tamanho das instâncias

O reduzido número de iterações observado na Tabela 8.158 para as instâncias com mais de 200 servidores desencoraja testes para estas instâncias visto que um número tão reduzido de iterações não é suficiente para fornecer conclusões definitivas. Devido a este fato, só foram geradas instâncias para a classe D com um número de servidores menor ou igual a 100, e portanto, todas as instâncias de 200, 300 e 400 servidores apresentadas nesta tese pertencem à classe C.

A Tabela 8.159 apresenta os *gaps* obtidos pelas duas versões do *ILS* para as instâncias de 300 servidores. Em alguns casos nestas instâncias também é possível observar que os

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
300C01*	0.000000	0.000000	**-0.000004**	-0.000001
300C02**	**-2.467219**	-2.463886	-2.462569	-2.462404
300C03	**-3.797414**	-3.797348	-3.797374	-3.797291
300C04	**-6.571241**	-6.571063	-6.571118	-6.571061
300C05	**0.000000**	0.000000	**0.000000**	0.000000

Tabela 8.159: Gaps: ILS X ILSAM - 300 Servidores

dois algoritmos conseguem atingir resultados melhores que os obtidos pelo método exato. note que a instâncias marcadas com "*" e com "**". As instâncias marcadas com "*" são instâncias onde o *ILSAM* apresenta resultados médios melhores que a melhor solução obtida pelo *ILS*. As instâncias marcadas com "**" são instâncias onde ocorre o contrário, ou seja, o algoritmo *ILS* atinge resultados médios melhores que a melhor solução obtida pelo *ILSAM*. Estes resultados ocorrem principalmente pelo fato de que para as instâncias com mais de 100 servidores o número de iterações feitas pelos algoritmos *ILS* e *ILSAM* é extremamente reduzido, o que por sua vez pode ocasionar este tipo de resultado uma vez que o algoritmo *ILSAM* precisa de um número razoável de iterações para que os mecanismos de intensificação e diversificação possam ser efetivos.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
300C01	2976.00	3565.67	6034.00	4416.67
300C02	3837.00	6925.00	5627.00	7762.33
300C03	7620.00	4626.67	3314.00	9237.67
300C04	10683.00	8843.67	8660.00	9578.67
300C05	3868.00	4688.33	3727.00	4652.33

Tabela 8.160: Tempos: ILS X ILSAM - 300 Servidores

A Tabela 8.160 mostra os tempos computacionais obtidos pelos algoritmos *ILS* e *ILSAM*. A alternância dos algoritmos no tempo para encontrar as melhores soluções novamente é observada. No entanto, esta alternância não é observada na média dos tempos, sendo que o algoritmo *ILS* possui as menores médias de tempo para todas as instâncias exceto a instância 300C05.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
400C01	**-2.4903**	-2.4717	-2.4902	-1.6601
400C02	**-0.000004**	-0.000001	**-0.000004**	-0.000003
400C03	-6.23728	-6.2372	**-6.23733**	-6.2373
400C04	-4.05106	-4.0410	**-4.05113**	-4.0411
400C05	**-9.5629**	-6.8032	-9.5628	-6.3740

Tabela 8.161: Gaps: ILS X ILSAM - 400 Servidores

A Tabela 8.161 mostra os gaps obtidos pelo *ILS* e pelo *ILSAM* para as instâncias de 400 servidores. O algoritmo *ILSAM* consegue os melhores resultados e as melhores médias para as instâncias 400C02, 400C03 e 400C04, enquanto o *ILS* obtém melhores

resultados para as instâncias 400C01 e 400C05.

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
400C01	10839.00	12997.33	17482.00	13184.67
400C02	14296.00	13040.00	15020.00	14951.00
400C03	9687.00	12373.00	11876.00	15360.33
400C04	9984.00	11507.67	9485.00	11537.67
400C05	13309.00	14266.33	22592.00	15802.67

Tabela 8.162: Tempos: ILS X ILSAM - 400 Servidores

A Tabela 8.162 mostra os tempos obtidos para as instâncias de 400 servidores pelos algoritmos *ILS* e *ILSAM*. De acordo com a tabela não é possível verificar a alternância entre os algoritmos para os melhores resultados e médias, sendo o algoritmo *ILS* o que apresenta as melhores médias em todas as instâncias e que apresenta tempo superior na obtenção da melhor solução em apenas uma instância.

Tendo em vista os resultados expostos para as instâncias com até 100 servidores pode-se concluir que o algoritmo *ILSAM* apresenta resultados melhores que o algoritmo *ILS* visto que em grande parte dos casos o *ILSAM* apresenta resultados médios com qualidade superior aos melhores resultados obtidos pelo *ILS*, e também que na grande maioria dos casos o *ILSAM* obteve médias melhores que as médias obtidas pelo *ILS*. No que tange os tempos computacionais, pode-se concluir que os algoritmos possuem comportamento muito similar uma vez os algoritmos se alternam no tempo de obtenção da melhor solução e também no tempo médio para obter os melhores resultados.

Os resultados para ambos os algoritmos para as instâncias de 200, 300 e 400 não podem ser considerados conclusivos devido ao reduzido número de iterações obtidos pelos dois algoritmos nestas instâncias.

8.12 Resultados adicionais

Esta seção apresenta resultados adicionais relevantes em relação às metodologias propostas.

8.12.1 Comparação entre os algoritmos *RTR* e o *RTR-Adaptativo*

A Tabela 8.163 mostra a comparação entre a qualidade das soluções obtidas pelos algoritmos *RTR* e *RTR-Adaptativo*. Na primeira coluna estão as instâncias utilizadas. Na segunda e terceira colunas, respectivamente, estão a melhor solução e a média das soluções encontradas pelo algoritmo *RTR* convencional. Já nas colunas quatro e cinco estão o valor

da função objetivo da melhor solução e a média das funções objetivo encontradas pelo *RTR-Adaptativo*. Os dois algoritmos foram executados com as mesmas sementes (5, 7 e 11) para cada instância e em todas as instâncias, os algoritmos tiveram a mesma solução como solução inicial.

	RTR		RTR-Adaptativo	
Instância	Melhor	Média	Melhor	Média
20D16*	18518762.00	18518762.00	18447855.30	18485193.63
20D17*	17610882.30	17610882.30	17602526.15	17605737.61
20D18	18169057.27	18171435.30	18169057.27	18171208.35
20D19*	17554794.18	17558752.97	17540645.05	17548960.34
20D20*	17758677.58	17758682.41	17738607.94	17751340.75
30D16	27132576.88	27148299.85	27131473.60	27141618.78
30D17*	26693102.41	26700367.71	26648541.19	26668971.80
30D18*	25900245.95	25901480.76	25874508.28	25890028.66
30D19*	27196961.64	27196961.64	27181264.25	27189863.03
30D20*	28141343.83	28141946.55	28127348.04	28133954.68

Tabela 8.163: *RTR* vs *RTR-Adaptativo*

É importante notar que em nenhuma das instâncias testadas o algoritmo *RTR-Adaptativo* obteve um resultado pior que o algoritmo *RTR* e em oito das dez instâncias testadas, marcadas com * na Tabela 8.163, a média da qualidade das soluções encontradas pelo *RTR-Adaptativo* tem qualidade superior à da melhor solução encontrada pelo *RTR* convencional.

	gap ILS (%)		gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
30C11*	3.643	3.954	**3.223**	3.479
30C12*	1.363	1.420	**1.085**	1.281
30C13	**0.998**	1.029	1.028	1.079
30C14	0.922	1.022	**0.865**	0.949
30C15	0.665	0.773	**0.558**	0.731
30D16*	9.372	9.520	**7.175**	7.231
30D17*	10.689	11.494	**8.957**	9.217
30D18*	9.296	9.498	**7.363**	7.577
30D19*	9.945	10.183	**7.792**	8.043
30D20*	9.893	9.978	**7.514**	7.730

Tabela 8.164: Gaps: ILS X ILSAM - 30 Servidores - Solução Inicial Aleatória

A Tabela 8.164 mostra os resultados obtidos pelos algoritmos *ILS* e *ILSAM* usando a mesma solução aleatória como ponto de partida. A diferença destes resultados para o apresentado anteriormente é que nestes experimentos o algoritmo construtivo *HNH-Futuro* não é utilizado. Para gerar a solução inicial neste experimento é utilizada uma heurística que constrói um posicionamento de replicas aleatório para cada período e usando este posicionamento aleatório PDR é resolvido através do modelo de fluxo em rede apresentado na Seção 6.2.4. É importante notar que, novamente, o *ILSAM* apresenta resultados médios com qualidade superior à melhor solução encontrada pelo *ILS* em grande parte dos casos. Outro ponto a ser destacado é mesmo após duas horas de execução partindo de uma solução aleatória, nenhum dos dois algoritmos consegue atingir o resultado obtido pela

heurística construtiva HNH-Futuro. Este fato mostra que uma boa heurística construtiva pode ter muita influência na qualidade final da soluções em algoritmos derivados da meta-heurística *ILS* para o PPRDR, assim como indicado por [22].

	Tempos ILS (s)		Tempos ILSAM (s)	
Instância	Melhor	Média	Melhor	Média
30C11	7218.00	7208.33	7104.00	7172.33
30C12	7213.00	7146.33	6909.72	7042.94
30C13	7184.00	7075.95	7207.00	7184.67
30C14	7155.00	7172.33	7183.00	7119.33
30C15	7124.00	7134.33	7083.00	7102.67
30D16	7119.00	7174.67	7262.00	7199.00
30D17	7188.00	7205.33	7255.00	7243.67
30D18	7237.00	7232.33	7213.00	7189.67
30D19	7342.00	7289.67	7220.00	7260.67
30D20	7217.00	7216.33	7000.00	7148.67

Tabela 8.165: Tempos: ILS X ILSAM - 30 Servidores - Solução Inicial Aleatória

A Tabela 8.165 mostra os tempos obtidos pelos Algoritmos *ILS* e *ILSAM* no o experimento com solução inicial aleatória. Também neste caso a alternância dos algoritmos para obter os melhores resultados pode ser observada.

Capítulo 9

Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões que podem ser tiradas de acordo com os resultados experimentais obtidos e algumas sugestões de trabalhos futuros.

9.1 Conclusões

Neste trabalho são apresentados uma descrição do Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR), presente em Redes de Distribuição de Conteúdos, bem como uma série de abordagens exatas para sua versão *offline* que incluem uma formulação matemática e um algoritmo de enumeração. Também para a versão *offline* são propostas duas versões da meta-heurística ILS, sendo uma delas adaptativa, que tenta escapar de ótimos locais através da modificação de seus próprios parâmetros. Também são propostas várias heurísticas para a versão *online*, baseadas na decomposição do PPRDR em dois subproblemas, o Problema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR). Para cada um destes subproblemas diferentes estratégias de resolução são utilizadas, criando assim um extenso conjunto de combinações de algoritmos para a resolução do problema.

O modelo do problema utilizado trata de maneira conjunta a replicação de conteúdos, posicionamento das réplicas e distribuição das requisições para arquivos extensos, analisando não apenas os custos para associar as requisições aos servidores, mas também uma série de questões que até então, segundo o conhecimento do autor, não haviam sido abordadas de maneira conjunta, como banda mínima para os clientes, carga nos servidores e presença de múltiplos conteúdos. Além dessas questões, uma série de outras, relacionadas ao fato dos conteúdos serem extensos, como possibilidade de atendimento por mais de um servidor ao mesmo tempo e a possibilidade de um atendimento se estender por vários

períodos de tempo, também são discutidas e analisadas.

Várias das heurísticas propostas são métodos híbridos, que conjugam conceitos de métodos exatos e heurísticos ao mesmo tempo. Todos as heurísticas híbridas foram testadas para um extenso conjunto de instâncias e os resultados mostram que as heurísticas híbridas conseguem atingir resultados muito bons, quando comparados com as melhores soluções obtidas para a versão *offline* do problema. Dentre as heurísticas para a versão *online*, destacam-se *HNH* e *HC* por seu desempenho para todas as instâncias em termos de qualidade de solução, as heurísticas *ALGA* e *ASPA* por seu desempenho em termos de qualidade de solução para as instâncias de grande porte, e a heurística *SPA*, por seu desempenho em termos de tempo computacional. No que tange os estimadores de demanda, destaque deve ser dado aos estimadores AEBH pelo seu desempenho em termos de qualidade de previsão e ao estimador Bote por seu desempenho em termos de qualidade de previsão e simplicidade.

A análise dos resultados também mostra que duas das quatro classes de instâncias propostas (A e B) são de fácil resolução e que as instâncias de uma das classes (C) gera PDRs de fácil resolução quando particionadas. A classe D de instâncias é de fato a mais difícil de se resolver entre as classes propostas visto que até para algumas instâncias de pequeno porte o resolvedor CPLEX apresenta dificuldades para encontrar as soluções ótimas.

Pode-se concluir também que é possível melhorar a eficiência da meta-heurística *ILS* e *RTR* sem que complexidade adicional seja introduzida nas iterações, visto que através de simples modificações no comportamento destas meta-heurísticas provocam melhorias consideráveis na qualidade das soluções produzidas. Também se pode concluir que o uso das meta-heurísticas *ILS*, tanto normal como adaptativa, não foi proveitoso para as instâncias com mais de 200 servidores visto que, no tempo limite estabelecido, estas meta-heurísticas não são capazes de executar um volume de trabalho significativo o suficiente para que os resultados possam ser considerados conclusivos.

9.2 Trabalhos Futuros

Visto que, como dito na Seção 9.1, apenas uma das classes de instâncias geradas pode ser considerada difícil, uma das frentes de trabalho a serem exploradas é a de confecção de novas instâncias para o PPRDR e para problemas similares como por exemplo o Problema de Dimensionamento da Capacidade de Servidores [48].

Como as versões da meta-heurística *ILS* não produziram resultados significativos para as instâncias com mais de 200 servidores, uma frente de trabalho que parece promissora é o uso de outras meta-heurísticas, como por exemplo, variações de Algoritmos Genéticos [24] e *GRASP* [20] para a versão *offline* do PPRDR.

Ainda se tratando da versão *offline*, o sucesso do algoritmo SEA para resolução de instâncias de grande porte e uma de pequeno porte motiva o aprimoramento deste algoritmo e uso de variações deste algoritmo para diferentes problemas onde os métodos exatos tradicionais apresentam dificuldade na resolução de instâncias.

Ainda como trabalho futuro pode-se citar a sugestão de um dos revisores do artigo [36], onde a inclusão de restrições na banda dos servidores para o download de conteúdos pode aproximar ainda mais o modelo tratado da realidade.

Referências

[1] AKAMAI. World Wide Web, www.akamai.com. 03/2008.

[2] Level 3. World Wide Web,www.level3.com. 01/2009.

[3] Mirror Image. World Wide Web, www.mirror-image.com. 01/2009.

[4] BRITE. World Wide Web, http://www.cs.bu.edu/brite/, June 2007.

[5] ILOG S.A., CPLEX 11 user's manual, 2008.

[6] GNU linear programing kit (glpk). www.gnu.org/software/glpk, 2010.

[7] Ahuja, R. K., Magnanti, T. L., Orlin, J. *Network Flows:Theory, Algorithms, and Applications*, 1 ed. Prentice Hall, 1993.

[8] Aioffi, W., Mateus, G., Almeida, J., Loureiro, A. Dynamic content distribution for mobile enterprise networks. *IEEE Journal on Selected Areas in Communications 23*, 10 (2005), 2022–2031.

[9] Almeida, J. M., Eager, D. L., Vernon, M. K., Wright, S. J. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE TRANSACTIONS ON MULTIMEDIA 6* (2004), 356–365.

[10] An, C., Fromm, H., Eds. *Supply Chain Management on Demand: Strategies and Technologies, Applications*. Springer, 2005.

[11] Arifovic, J. Genetic algorithm learning and the cobweb model. *Journal of Economic Dynamics and Control* (1994).

[12] Bakiras, S., Loukopoulos, T. Combining replica placement and caching techniques in content distribution networks. *Computer Communications 28* (2005), 1062–1073.

[13] Bartolini, N., Presti, F., Petrioli, C. Optimal dynamic replica placement in content delivery networks. In *The 11th IEEE International Conference on Networks 2003. ICON2003* (2003), p. 125–130.

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

[15] Besten, M. D., Stützle, T., Dorigo, M. Design of Iterated Local Search Algorithms. In *Applications of Evolutionary Computing. EvoWorkshops2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM. Proceedings* (Como, Italy, 2001), E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. Raidl, R. E. Smith, and H. Tijink, Eds., vol. 2037, Springer-Verlag, p. 441–451.

[16] Breslau, L., Cao, P., Phillips, G., Shenker, S. Web Caching and Zipf-like Distribuitions: Evidence and Implications. In *Proceedings of IEEE INFOCOM 99* (1999), p. 126–134.

[17] Buriol, L., Resende, M., Thorup, M. Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing 20* (2008).

[18] Coppens, J., Wauters, T., Turck, F. D., Dhoedt, B., Demeester, P. Design and performance of a self-organizing adaptive content distribution network. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* (2006), p. 534–545.

[19] Dias, J., Captivo, M. E., Clímaco, J. Capacitated dynamic location problems with opening, closure and reopening of facilities. *Journal of Managment Mathematics 17* (2006), 317–348.

[20] Festa, P., Resende, M. GRASP: An annotated bibliography. In *Essays and surveys in metaheuristics*, C. Ribeiro and P. Hansen, Eds. Kluwer Academic Publishers, 2002, p. 325–367.

[21] Galvão, R. D., acosta Espejo, L. G., Boffey, B. A hierarchical model for the location of perinatal facilities in the municipality of rio de janeiro. *European Journal Of Operational Research 138* (2002), 495–517.

[22] Glover, F., Kochenberger, G., Eds. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2002.

[23] Glover, F., Laguna, M. *Tabu Search*. Kluwer Academic Publishers, 1997.

[24] Holland, J. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[25] Huang, C., Abdelzaher, T. Towards content distribution networks with latency guarantees. In *Twelfth IEEE International Workshop on Quality of Service, 2004. IWQOS 2004* (2004), p. 181–192.

[26] Kirkpatrick, S., Gellat, D. C., Vecchi, M. P. Optimization by simulated annealing. *Science 220* (1983), 671–680.

[27] Korupolo, M. R., Plaxton, C. G. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms 37* (2000), 146–188.

[28] Kurose, J., Ross, K. *Computer Networking: a Top-Down Approach Featuring the Internet*. Addison-Wesley, 2003.

[29] Leighton, F. T., Lewin, D. Global hosting system, August 2000. US Patent:US006108703.

[30] Li, F., Golden, B., Wasil, E. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers and Operations Research 32* (2005), 1165–1179.

[31] Li, F., Golden, B., Wasil, E. A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Computers and Operations Research 34* (2007), 2734–2742.

[32] Maier, H. R., Dandy, G. C. Neural networks for the prediction and forecasting of water resources variables: a review of modelling issues and applications. *Environmental Modelling & Software 15* (2000), 101–124.

[33] Martello, S., Toth, P. *Knapsack Problems.* John Wiley & Sons, 1990.

[34] Morettin, P. A., de Castro Toloi, C. M. *Modelos para Previsão de Séries Temporais*, vol. 1. Instituto de Matemática Pura e Aplicada, 1981.

[35] Neves, T. longpaper. Relatório Técnico, UFF, 2010.

[36] 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.

[37] Neves, T., Drummond, L., Uchoa, E., Albuquerque, C. Otimização em redes de distribuição de conteúdos. In *Anais do XL Simpósio Brasileiro de Pesquisa Operacional* (2008), p. 2345–2356.

[38] Neves, T., Drummond, L., Uchoa, E., Ochi, L., Albuquerque, C. Replicação e distribuição *Online* em redes de distribuição de conteúdos. In *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional* (2009), p. 2717–2727.

[39] Neves, T., Ochi, L., Drummond, L., Uchoa, E., Albuquerque, C. Optimization in content distribution networks. In *Proc. of the International Conference on Engineering Optimization (EngOpt2008)* (2008).

[40] Neves, T. A. Heurísticas com memória adaptativa aplicadas ao problema de roteamento e scheduling de sondas de manutenção. Dissertação de Mestrado, Universidade Federal Fluminense - UFF, 2007.

[41] Pirkul, H., Jayaraman, V. A multi-commodity, multi-plant capacitated facility location problem: Formulation and efficient heuristic solution. *Computers & Operations Research 25*, 10 (1998), 869–878.

[42] Silberschatz, A., Galvin, P., Gagne, G. *Operating System Concepts with Java*, 7 ed. Jhon Wiley and Sons. Inc, 2007.

[43] Taillard, E., Gambardella, L., Gendreau, M., Potvin, J.-Y. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operation Research 135* (2001), 1–16.

[44] Talbi, E.-G. *Metaheuristics: From Design to Implementation.* John Wiley & Sons, 2009.

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

[46] Tenzakhti, F., Day, K., Ould-Khaoua, M. Replication algorithms for the world-wide web. *Journal of Systems Architecture 50* (2004), 591–605.

[47] Toso, R. Algoritmos para atualização de árvores geradoras mínimas em grafos dinâmicos. Dissertação de Mestrado, Universidade Federal Fluminense - UFF, 2006.

[48] Uderman, F., Neves, T., Albuquerque, C. Optimizing server storage capacity on content distribution networks. In *Anais do Simósio Brasileiro de Redes de Computadores - SBRC2011. to apear* (2011).

[49] Wauters, T., Coppens, J., Dhoedt, B., Demeester, P. Load balancing through efficient distributed content placement. In *Next Generation Internet Networks* (2005), p. 99–105.

[50] Wolfson, O., Jajodia, S., Huang, Y. An adaptive data replication algorithm. *ACM Transactions on Database Systems 22*, 2 (1997), 255–314.

[51] Zhang, G., Patuwo, B. E., Hu, M. Y. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting 14*, 1 (1998), 35–62.

[52] Zhou, X., Xu, C.-Z. Efficient algorithms of video replication and placement on a cluster of streaming servers. *Journal of Network and Computer Applications 30* (2007), 515–540.

[53] Zionts, S. *Linear and Integer Programming*. Prentice-Hall, 1974.

[54] Zipf, G. K. *Relative Frequency as a Determinant of Phonetic Change*, vol. 40 of *Harvard Studies Classical Philology*. Harvard University Press, 1929.