UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

POPMUSIC APLICADA AO PROBLEMA DE POSICIONAMENTO DE RÉPLICAS E DE DISTRIBUIÇÃO DE REQUISIÇÕES EM REDES DE DISTRIBUIÇÃO DE CONTEÚDOS

Ronaldo Goldbach

Orientadores

Adriana Cesário de Faria Alvim
Tiago Araújo Neves

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2013

POPMUSIC APLICADA AO PROBLEMA DE POSICIONAMENTO DE RÉPLICAS E DE DISTRIBUIÇÃO DE REQUISIÇÕES EM REDES DE DISTRIBUIÇÃO DE CONTEÚDOS

Ronaldo Goldbach

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

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

Prof. Tiago Araújo Neves, D.Sc. - UFF

Prof. Morganna Carmem Diniz, D.Sc. - UNIRIO

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

Prof. Eduardo Uchoa Barboza, D.Sc.- UFF

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2013

Goldbach, Ronaldo.
G618 Popmusic aplicada ao problema de posicionamento de réplicas e de distribuição de requisições em redes de distribuição de conteúdos / Ronaldo Goldbach, 2013.
iv, 69 f. ; 30 cm

Orientadora: Adriana Cesário de Faria Alvim.
Coorientador: Tiago Araújo Neves.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

1. Heurística. 2. Metaheurística. 3. POPMUSIC. 4. Redes de Distribuição de Conteúdo. I. Alvim, Adriana Cesário de Faria. II. Neves, Tiago Araújo. III. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. IV. Título.

CDD – 004

DEDICATÓRIA

Dedico este trabalho à minha família.

À minha esposa, Aida, pelo incentivo desde o início e durante todo o percurso, pelo suporte, pela compreensão em momentos de maior estresse, pelo companheirismo quando foram inevitáveis restrições que tive que escolher na vida social, por assumir tarefas que eu não tinha como cumprir – enfim, por seu amor imenso e irrestrito.

Às minhas filhas, esperando que os exemplos de esforço contínuo para aperfeiçoamento, de nunca querer parar, de nunca considerar idade ou capacidade um problema, lhes ajude a vencer em situações que a vida eventualmente impõe. Tenham a certeza de sempre contar comigo.

Aos meus pais, que desde sempre me deram apoio e exemplo, incentivo para aprender e me desenvolver, que me desafiaram e cobraram. Sempre houve um alto nível de exigência, ao mesmo tempo em que sempre houve muito carinho e compreensão. Isso fez com que eu desenvolvesse minha curiosidade, expandisse meus horizontes, enfrentasse e superasse o que poderiam ter sido limites. Meu pai... Sinto tanto sua falta! Tenho certeza de que você também está orgulhoso.

AGRADECIMENTOS

Aos meus incansáveis orientadores, Adriana e Tiago, que nunca esmoreceram no propósito de me fazer apresentar o melhor trabalho possível. Sem a dedicação de ambos, este trabalho não existiria.

Aos meus professores da UNIRIO, em especial Ângelo, Pimentel e Sean, que me incentivaram a pensar e enxergar de forma diferente e, me desafiaram a tentar entender temas que me eram completamente novos. Confesso que ainda não entendi alguns.

Aos meus colegas de curso, pelo apoio e ajuda nos momentos em que a soma da carga de trabalho discente com as obrigações profissionais pareceu uma equação impossível de ser resolvida.

Aos meus superiores na Petrobras, por acreditar na minha capacidade, pelo apoio e incentivo recebidos ao trilhar este caminho. Eles, assim como meus colegas de trabalho, muitas vezes compreenderam e compensaram minhas eventuais ausências físicas ou mentais...

Aos meus amigos, que aceitaram o distanciamento que tive que impor e ao mesmo tempo não se afastaram de todo, mantendo contato e os laços não só intactos mas reforçados.

A Deus, por ter colocado tantas boas pessoas no meu caminho.

GOLDBACH, Ronaldo. **POPMUSIC aplicada ao Problema de Posicionamento de Réplicas e de Distribuição de Requisições em Redes de Distribuição de Conteúdos**. UNIRIO, 2013. 73 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

O tamanho e a demanda por conteúdos digitais tem aumentado ao longo dos últimos anos, para fins de entretenimento, educação e uso profissional. Um modo de atender a demanda é o emprego de Redes de Distribuição de Conteúdos (RDCs), compostas por computadores (ou "servidores") conectados através da internet. No entanto, não se sabe *a priori* quais conteúdos serão requisitados, nem quando; e por limitações de ordem física e econômica, não é viável a replicação de todos os conteúdos em todos os servidores. Para atender às necessidades dos clientes e manter os custos, incluindo os relativos a tráfego na rede, no nível mais baixo possível, é preciso decidir a cada período de tempo em que servidores posicionar réplicas de quais conteúdos, e a que servidor direcionar cada requisição feita. Os diversos arranjos possíveis caracterizam problemas combinatórios para os quais é muito difícil obter soluções de boa qualidade num tempo computacional razoável, em instâncias de tamanho "grande". Este trabalho desenvolve e implementa um algoritmo, POP-PPRDR, que aplica a técnica POPMUSIC (*Partial OPtimization Metaheuristic Under Special Intensification Conditions*) ao Problema de Posicionamento de Réplicas e de Distribuição de Requisições (PPRDR) em RDCs. POPMUSIC decompõe problemas em subproblemas menores para então tratá-los. Testes do POP-PPRDR obtiveram soluções de melhor qualidade do que soluções obtidas por método testado recentemente para resolver o PPRDR.

Palavras-chave: Heurística, Metaheurística, POPMUSIC, Posicionamento de Réplicas, Distribuição de Requisições, PPRDR, RDC

ABSTRACT

Both size of, and demand for digital contents, have been growing through the last years, to entertainment, education and professional purposes. The use of Content Distribution Networks (CDNs), comprised of internet-connected computers (or "servers"), is one possible answer to this demand. Nevertheless, one does not know *a priori* what will be the demand, nor when it will happen; and it is not physically nor economically viable replicate every content in every server. To cope with customers' requirements and keep costs, including traffic costs, as low as possible, it is important to decide for each period of time what content will be kept on what server, and what request will be directed to what server. The different possible arrangements characterize a combinatorial problem to which a good-quality computer-based solution is very hard to reach in a reasonable time, when dealing with "big"-sized instances. This work develops and implements an algorithm, POP-PPRDR, that applies POPMUSIC (*Partial OPtimization Metaheuristic Under Special Intensification Conditions*) to the Replica Placement and Requisition Distribution Problem (RPRDP) in CDNs. POPMUSIC decomposes problems in smaller problems and then try to solve them. POP-PPRDR testing obtained better quality solutions when compared against solutions obtained by a recently tested method used to solve RPRDP.

Keywords: Heuristics, Metaheuristics, POPMUSIC, Replica Placement, Requisition Distribution, RPRDP, CDN

Sumário

1. Introdução

Quando uma pessoa deseja usar a internet para ver um videoclipe no seu computador, ou baixar uma versão de um programa antivírus, ela está gerando uma demanda pelo que se chama neste trabalho de "serviço de *download*" (ou apenas *download*) de um "conteúdo". As demandas, muitas vezes, não são previsíveis e podem coincidir, no tempo, com demandas de outras pessoas ("clientes") que pretendem usar serviços de *download* a partir de um mesmo computador onde o conteúdo desejado fica armazenado. Em alguns casos, o computador, também chamado "servidor", pode ficar sobrecarregado: sua "banda" (o volume máximo de dados transmissível, a cada momento, pela ligação entre um servidor e seus clientes) pode ser insuficiente para atender simultaneamente a todas as requisições feitas.

De acordo com BAKIRAS e LOUKOPOULOS (2005), TENZAKHTI *et al.*, (2004), e ZHOU e XU, (2007), o problema apresentado no parágrafo acima pode ser modelado pelo uso de Redes de Distribuição de Conteúdos (RDC – em inglês, CDN, *Content Distribution Network*), formadas por servidores virtuais hospedados em servidores de topologia real. Uma RDC hospeda e distribui o conteúdo demandado. Manter uma réplica de cada um dos conteúdos em todos os servidores garantiria ter qualquer conteúdo o mais perto possível de cada possível cliente, mas na vida real isso pode ser muito caro ou até impossível: há restrições na capacidade de armazenamento e na banda de cada servidor, na quantidade máxima permitida de cópias de certos conteúdos etc. Uma RDC posiciona uma ou mais réplicas de conteúdos em alguns servidores para equilibrar a carga entre eles, minimizar o tráfego e, portanto, os congestionamentos; e evitar, ou ao menos reduzir, atrasos na entrega dos conteúdos requisitados. Às vezes, reduzir atrasos significa atender a uma requisição a partir de um servidor mais distante do cliente, porém com mais banda disponível. A entrega de alguns conteúdos demandados exige um nível mínimo de "Qualidade de Serviço" (QoS

– *Quality of Service*) a certos clientes, estipulado em contrato – por exemplo na forma de um tempo máximo de atendimento, ou como uma banda mínima que tem que ser disponibilizada.

Muitos estudos têm sido realizados à procura de melhores formas de minimizar o tráfego na rede, respeitando restrições como QoS, largura de banda e capacidade de armazenamento de servidores (NEVES *et al.*, 2008, NEVES, 2011). Entre os diversos problemas de gerenciamento em uma RDC é possível citar:

a) O Problema de Localização de Servidores (PLS): consiste em localizar, sobre os servidores da rede real, cada servidor da estrutura virtual, com o objetivo de minimizar o tráfego na rede (BEKTAS *et al.*, 2007);

b) O Problema de Replicação (PR): consiste em decidir que conteúdos replicar e em que quantidade (ZHOU e XU, 2007);

c) O Problema de Posicionamento de Réplicas (PPR): consiste em decidir em que servidor(es) posicionar cada réplica, com o objetivo de minimizar o tráfego e os custos de transmissão (NEVES, 2011, ZHOU e XU, 2007). A solução para o PPR será tanto melhor quanto mais precisa for a previsão de requisições. Os dados de Posicionamento de Réplicas são uma entrada fundamental para o próximo problema;

d) O Problema de Distribuição de Requisições (PDR): consiste em distribuir a demanda entre os servidores, uma vez as réplicas posicionadas, com o objetivo de minimizar os custos de transmissão.

O PPRDR, Problema de Posicionamento de Réplicas e Distribuição de Requisições, tratado nos trabalhos de NEVES *et al.* (2008) e NEVES (2011), é uma variante do PPR, na qual PPR e PDR devem ser resolvidos em conjunto, alocando cópias do conteúdo próximo de onde se prevê que sejam requisitados e dirigindo as requisições que ocorrem ao(s) servidor(es) mais próximo(s) do cliente (ou com mais banda disponível) que hospede(m) uma réplica do conteúdo solicitado. O objetivo é reduzir ou eliminar atrasos na entrega, procurando respeitar as exigências de qualidade de cada cliente/requisição. O PPRDR pertence à classe NP-Difícil (GAREY e JOHNSON, 1979, NEVES, 2011), cujos problemas têm sido tratados por dois tipos de métodos: exatos e heurísticos. Enquanto os primeiros garantem alcançar a solução ótima para o problema, os métodos heurísticos não oferecem essa garantia; possibilitam,

porém, o tratamento de instâncias maiores em tempo menor (REEVES e BEASLEY, 1993).

NEVES (2011) apresenta diversas técnicas de solução para o problema, entre elas uma que emprega métodos heurísticos e meta-heurísticos e que obteve resultados excelentes, quando comparados a outros resultados existentes na literatura, para instâncias com até 100 servidores. A mesma técnica, no entanto, não chega a uma conclusão em termos de qualidade da solução para instâncias maiores, considerando o critério de parada utilizado que limita a execução do algoritmo em um determinado tempo. Como trabalhos futuros, o autor propõe a utilização de diferentes metaheurísticas para resolver o PPRDR. No entanto, mesmo considerando a escalabilidade das metaheurísticas com o tamanho dos problemas, a partir de certo porte a solução pode deixar de ser eficiente – o limite de resolução prático de muitos algoritmos é de instâncias com algumas poucas centenas de itens. Instâncias de problemas combinatórios reais que podem envolver milhares de itens estão claramente além desse limite e precisam ser resolvidas de outra maneira (OSTERTAG *et al.*, 2009). Uma das maneiras é a decomposição do problema em problemas de menor porte (TAILLARD *et al.*, 2007).

O objetivo específico desta dissertação de mestrado é propor uma heurística capaz de prover solução para o PPRDR em RDCs de médio e grande portes. Entre outras características, deseja-se que o algoritmo proposto seja robusto, isto é, estável em relação a diferentes instâncias do problema. Para isso, será empregada a metaheurística POPMUSIC (*Partial OPtimization Method Under Special Intensification Conditions*), especialmente eficiente para resolver problemas combinatórios de larga escala que possam ser otimizados por partes (TAILLARD e VOSS, 2001). POPMUSIC parte de uma solução para o problema original, não necessariamente de boa qualidade, monta um subproblema e o disponibiliza para tratamento em busca de um ótimo local. POPMUSIC controla, ainda, a aplicação – no problema completo – das melhorias eventualmente encontradas durante o tratamento do subproblema. O procedimento continua – um novo subproblema é montado, tratado e eventuais melhorias incorporadas ao problema completo – até que não haja mais subproblemas a melhorar. Neste trabalho, a técnica empregada por NEVES *et al.* (2008) e NEVES (2011) com sucesso para resolver o PPRDR foi adaptada para ser utilizada no tratamento dos subproblemas criados pelo método POPMUSIC.

O trabalho está organizado da seguinte forma: o Capítulo 2 apresenta conceitos que permitem ao leitor um melhor entendimento do restante do texto, uma revisão de trabalhos relacionados e o método de decomposição POPMUSIC. O Capítulo 3 descreve o problema específico de gerenciamento de RDCs que é objeto de estudo da presente dissertação e detalha a solução proposta para o problema. O Capítulo 4 relata os experimentos computacionais realizados. Finalmente, no Capítulo 5 são apresentadas as conclusões do trabalho e sugestões para próximos passos da pesquisa.

2. Revisão Bibliográfica

Este capítulo é composto por duas seções. A primeira resume os principais conceitos teóricos que são abordados no trabalho, com base em três referências principais: HOOS e STÜTZLE (2005), REEVES e BEASLEY (1993) e TAILLARD e VOSS (2001). A segunda apresenta uma seleção de trabalhos que tratam de temas ligados ao gerenciamento e à otimização dos serviços prestados por Redes de Distribuição de Conteúdo, as RDCs, bem como alguns dos trabalhos disponíveis na literatura que tratam da metaheurística POPMUSIC e de sua aplicação.

2.1 Conceitos teóricos

O objetivo desta seção é fornecer uma breve visão dos principais conceitos teóricos que são abordados no restante do trabalho. Mais detalhes podem ser obtidos na literatura citada.

O PPRDR, tratado neste trabalho, é um problema combinatório. Problemas combinatórios aparecem em diversas áreas da ciência da computação e em outras disciplinas onde métodos computacionais são aplicados. De acordo com HOOS e STÜTZLE (2005) e REEVES e BEASLEY (1993), os problemas combinatórios envolvem encontrar um agrupamento, uma ordenação, ou uma atribuição de um conjunto discreto e finito de objetos que satisfaz dadas condições, entre um conjunto de possíveis alternativas. O mesmo problema pode ter mais do que uma solução. Para a maioria dos problemas combinatórios, o "espaço de soluções candidatas" é o conjunto de potenciais soluções para uma dada instância do problema. Esse conjunto é pelo menos exponencial no tamanho da instância e pode conter soluções inviáveis, que não atendem a todas as condições necessárias a uma solução, e soluções viáveis, que atendem todas as condições necessárias. Problemas combinatórios podem ser de decisão (determinar se há ou não há solução) ou de otimização (encontrar uma solução para a

instância com valor ótimo). Os problemas de otimização lidam com uma "função objetivo", que mede a qualidade da solução – o resultado gerado pela aplicação de dada solução a uma instância de um problema. Os mesmos autores ressaltam que muitos dos problemas combinatórios relevantes são NP-difíceis, não solucionáveis de forma eficiente até hoje.

Em um exemplo de problemas de atribuição citado por REEVES e BEASLEY (1993), *n* pessoas devem executar *m* tarefas, sendo c_{ij} o custo da pessoa *i* executar a tarefa *j* e sendo o objetivo minimizar o custo total da execução das tarefas. A solução ótima é uma permutação $\{t_1, ..., t_m\}$ de tarefas t_j atribuídas a cada pessoa *i*, para *i* variando de 1 a *n*, que minimize o custo total $\sum_{i=1}^{n} c_{it_i}$. Qualquer permutação é uma solução candidata. Qualquer permutação cuja função objetivo produza custo igual ao mínimo é, também, uma solução ótima.

De acordo com os mesmos autores, a vizinhança da solução de um problema pode ser definida como o conjunto de soluções que pode ser alcançado a partir de tal solução, através de operações simples, como por exemplo: remover, incluir ou trocar de posição componentes da solução. Se uma solução para um problema é melhor do que qualquer outra solução em sua vizinhança, então esta solução é o ótimo local de sua vizinhança. Um ótimo global pode ser definido como uma solução ótima para um problema combinatório. Pode haver mais do que uma solução ótima para o mesmo problema.

Algoritmos de busca local geralmente trabalham da seguinte maneira: para uma dada instância de um problema combinatório, a busca por soluções ocorre no espaço de soluções candidatas. Um algoritmo do gênero parte de uma solução inicial, e prossegue movendo-se iterativamente de uma solução candidata para outra, uma solução vizinha, sendo cada movimento baseado apenas no conhecimento local (HOOS e STÜTZLE, 2005).

Conforme os mesmos autores, uma perturbação é uma operação que pode gerar uma nova solução modificando os componentes de uma solução existente. A 'força' da perturbação pode ser entendida como equivalente à quantidade de componentes modificados. Perturbações desempenham um papel crítico nos algoritmos de busca; por exemplo, perturbações fracas normalmente levam a fases de busca relativamente curtas,

pois o mecanismo de busca local tende a alcançar o ótimo local em poucos passos; por outro lado, se fraca demais, a perturbação frequentemente fará com que a busca local subsequente descubra o mesmo ótimo local recém visitado, levando o algoritmo à estagnação. Uma política de perturbações que resulte na exploração de um espaço de soluções candidatas mais fracamente relacionadas com uma solução atual, por algum critério (valor, semelhança, distância etc.), tenta evitar que a busca fique limitada a espaços relativamente confinados, diversificando as vizinhanças examinadas. Uma política de perturbações que resulte na exploração de um espaço de soluções candidatas mais fortemente relacionadas com uma solução atual, ao contrário, busca melhorar a solução usando como guia, por exemplo, o valor da função objetivo, intensificando o exame da vizinhança onde se encontre a solução atual. Mecanismos de perturbação podem usar memória para, por exemplo, ajustar a força e o tipo de perturbação.

Este trabalho utiliza métodos de resolução heurística para problemas de otimização combinatória. A palavra "heurística" deriva do grego *heuriskein* que significa descobrir; "envolvido em, ou servindo como ajuda para, aprender a descobrir ou resolver problemas por métodos experimentais e, em especial, tentativa-e-erro" (MERRIAM-WEBSTER Dictionary). No contexto da otimização combinatória, o termo "método heurístico" é usado em contraste a métodos que garantem descobrir um ótimo global para um problema. A heurística é uma técnica que busca soluções boas, a um custo computacional razoável, sem ser capaz de garantir que sejam soluções ótimas, e em muitos casos nem mesmo sendo capaz de estabelecer quão perto da solução ótima uma dada solução está.

O que justifica o uso de métodos heurísticos? De acordo com REEVES e BEASLEY (1993), uma forma simples de tentar resolver uma instância de dado problema combinatório é listar suas soluções viáveis, avaliar a função objetivo de cada uma delas e escolher a melhor solução para o caso. Esse enfoque é conhecido como "enumeração completa" (por considerar cada uma de todas as soluções individuais) e é teoricamente possível, mas na prática não funciona para problemas combinatórios de tamanho razoável, por causa da quantidade de soluções possíveis. Um método heurístico, que alcance uma solução boa num tempo razoável, pode ser melhor do que a alternativa de solução ótima em um tempo inaceitavelmente longo. Além disso, deve-se considerar que para tratar um problema é necessário criar um modelo do mesmo, e considerar as dificuldades de se chegar a um modelo perfeito de um problema real;

quanto mais complexo o problema real, maior a dificuldade de modelá-lo. As heurísticas são mais flexíveis e capazes de lidar com funções objetivo e restrições mais complexas e realísticas do que os algoritmos exatos, de forma geral, conseguem. O emprego de heurísticas costuma possibilitar a criação de um modelo mais próximo do real e permite achar uma solução (ainda que aproximada) para tal modelo. Isso pode gerar um resultado de melhor qualidade do que achar uma solução exata para um modelo menos próximo do real.

Metaheurísticas, segundo GLOVER e KOCHENBERGER (2003), são métodos que orquestram uma interação entre procedimentos de melhoria local e estratégias de mais alto nível para criar processos capazes de escapar de ótimos locais de baixa qualidade e executar uma busca robusta num espaço de soluções. Metaheurísticas são usadas para achar soluções de alta qualidade para um número crescente de problemas complexos e de difícil definição, em particular, problemas combinatórios. Exemplos conhecidos de metaheurísticas incluem, de acordo com GENDREAU e POTVIN (2010), a Busca Tabu (em inglês *Taboo Search*, TS) (GLOVER, 1986) que tenta imitar processos inteligentes, provendo métodos que fazem uso de memória para evitar retrocessos; a *Simulated Annealing*, (SA) (KIRKPATRICK *et al.*, 1983) que usa uma analogia com processos termodinâmicos para montar um arcabouço de otimização de sistemas muito grandes e complexos (*Simulated Annealing* significa Resfriamento Lento Simulado, mas o nome do algoritmo é normalmente usado em inglês, mesmo na literatura em português); a Otimização de Colônias de Formigas (*Ant Colony Optimization*) (DORIGO *et al.*, 1991); os Algoritmos Genéticos (*Genetic Algorithms*) (HOLLAND, 1992), e a Busca em Vizinhança Variável (VNS – *Variable Neighborhood Search*) (MLADENOVIC e HANSEN, 1997).

Outra metaheurística importante e utilizada no presente trabalho é a Busca Local Iterada, conhecida na literatura por sua sigla em inglês, ILS (*Iterated Local Search*). Para evitar que um procedimento de busca pare prematuramente em um ótimo local, a ILS usa alternadamente dois passos de busca, um para alcançar o ótimo local da maneira mais eficiente possível, e o outro para escapar do ótimo local (HOOS e STÜTZLE, 2005).

Uma definição possível para um algoritmo que implementa uma ILS está representada na Figura 1. No exemplo, a solução inicial gerada é s_O (linha 1). Na linha

2, a busca local em s_0 retorna um ótimo local *s**. Enquanto não se atinge o critério de parada especificado (linha 7), uma perturbação em *s** considerando o *histórico* da busca gera *s'* (linha 4), e a busca local em *s'* gera *s*'* (linha 5). Na linha 6, a solução *s**, da qual o mecanismo de busca vai prosseguir, passa a ser a melhor solução entre *s** e *s*'*, considerando o *histórico.*

Iterated Local Search

1: s_0 ← GeraSoluçãoInicial

2: *s** ← BuscaLocal (s_0)

3: **repete:**

4: *s'* ← Perturbação (*s*, histórico*)

5: *s*'* ← BuscaLocal (*s'*)

6: *s** ← CritérioAceitação (*s*, s*', histórico*)

7: **até:** critério de parada alcançado

Figura 1. Pseudocódigo geral do método ILS baseado em HOOS e STÜTZLE (2005).

Para maximizar o desempenho de uma heurística baseada no método ILS, considerando a qualidade da solução e o tempo computacional, é necessário fazer a melhor escolha possível para seus componentes. Por causa das interações entre os componentes, esse é um problema difícil que em muitos casos tem que ser resolvido por um método heurístico (NEVES, 2011). Cada componente é explicado a seguir de acordo com HOOS e STÜTZLE (2005):

- solução inicial: a solução inicial, por definir de onde se começará a busca, tem influência na quantidade total de iterações até um resultado aceitável (e, portanto, no tempo computacional);

- busca local: a efetividade da busca local influencia de forma direta o desempenho do algoritmo;

- perturbação: o papel da perturbação é o de mudar a solução candidata corrente. A alteração não deve ser desfeita pelo próximo passo da busca local. Uma perturbação deve permitir à ILS escapar de um ótimo local para que a próxima fase tenha a chance de descobrir um novo ótimo local. Ao mesmo tempo, não deve impor uma diversificação forte demais, evitando as desvantagens de um recomeço completamente aleatório e reconstrução da solução a partir do zero;

- critério de aceitação: o critério de aceitação pode ser usado para controlar o equilíbrio da busca entre diversificação (quando se aceita $s*'$ independentemente da qualidade de sua solução) e intensificação (quando se aceita apenas a melhor solução entre $s*$ e $s*'$).

2.2 Trabalhos relacionados

O objetivo desta seção é apresentar uma seleção de trabalhos existentes na literatura que tratam de temas ligados ao gerenciamento e à otimização dos serviços prestados por Redes de Distribuição de Conteúdo, as RDCs. Entre esses trabalhos há estudos e propostas de solução para dois problemas distintos, embora fortemente relacionados: o Problema de Posicionamento de Réplicas (PPR) que consiste em decidir quais réplicas colocar em cada servidor numa RDC, e o Problema de Distribuição de Requisições (PDR) que consiste em definir quais servidores devem atender cada requisição dos clientes. São apresentados, também, trabalhos que utilizam a metaheurística POPMUSIC e sua aplicação em diferentes problemas.

2.2.1 Trabalhos relacionados aos Problemas PPR, PDR e PPRDR

Das soluções propostas para o PPR na literatura, a maioria trata o caso estático, cujo objetivo é achar o melhor posicionamento das réplicas a partir de um padrão de tráfego de requisições conhecido. Embora não reflita a dinâmica da maior parte dos casos reais, essa proposta pode ser usada em casos onde as requisições apresentam pequena ou nenhuma variação, ou para estudar o comportamento de algoritmos (VICARI, 2008). Uma variação do PPR estático é tratada por ALMEIDA *et al.* (2004). Os autores propõem o uso de uma árvore de *multicast* para entrega escalável de conteúdos aos clientes. De acordo com GASPARY (1996), diferentemente do *broadcast*, em que todos os conteúdos são transmitidos a todos, o *multicast* ou difusão seletiva permite que cada máquina escolha se deseja participar de uma transmissão. O servidor envia o conteúdo apenas uma vez e todos os clientes interessados recebem os dados. ALMEIDA *et al.* (2004) argumentam que os protocolos *multicast* desenhados até então não resolviam a definição de sistemas de distribuição de conteúdo escaláveis, que colocassem servidores de réplicas mais próximos de diversas populações de clientes e distribuíssem as requisições dos clientes e os fluxos de resposta de maneira a minimizar o custo total de servidores e rede, por duas razões. Em primeiro lugar, a proximidade do servidor e o

roteamento pelo caminho mínimo não garantem uso ótimo da largura de banda; em segundo lugar, o uso da banda pelo servidor aumenta com o número de réplicas. Para problemas escaláveis de fluxo de transmissão, o uso de fluxos *unicast* ou ponto-a-ponto, onde um servidor transmite dados para outro servidor específico (GASPARY, 1996), produz soluções que não são ótimos globais, já que a banda dos servidores tem que ser suficiente à transmissão multiplicada pela quantidade de clientes: um fluxo *unicast* da origem para cada destino. A solução proposta mostra que o posicionamento de réplicas e o problema de otimização de rotas podem ser expressos de forma simples. Os autores propõem um algoritmo usando uma formulação matemática e várias heurísticas para o problema. As heurísticas geram soluções quase-ótimas, todas a uma distância menor do que 16% da solução ótima, com custo computacional muito menor do que as soluções obtidas pelo algoritmo que emprega formulação matemática, mantendo o mesmo padrão de qualidade deste último. Os algoritmos usados apresentam complexidade para a solução total equivalente à aplicação do algoritmo de caminho mínimo de Dijkstra a cada cliente e cada possível servidor na rede. O trabalho aponta a necessidade de testes em redes maiores e mais diversas. Os autores consideram o custo da entrega de um conteúdo pela RDC, mas não tratam a qualidade percebida pelo cliente (Quality of Service – QoS). Outra limitação do trabalho é não levar em conta a otimização do posicionamento e roteamento para entrega de múltiplos conteúdos.

O posicionamento dinâmico de réplicas, considerando a variação na demanda dos usuários e o desempenho da rede, é tratado por BARTOLINI *et al.* (2003). A abordagem proposta modela o problema como um processo de decisão de Markov, capaz de lidar dinamicamente com condições de tráfego, nível de satisfação do usuário, custo para instalar, manter e remover réplicas em função da demanda, capacidade de armazenamento e carga dos servidores, usando distribuições matemáticas. Partindo de um posicionamento inicial das réplicas, que é igual ao posicionamento final do período anterior, o problema pode ser definido como sendo o de encontrar a melhor mudança a fazer no posicionamento para atender, da melhor maneira possível, à demanda do período atual. A partir da análise do comportamento das soluções ótimas encontradas pelo processo de Markov, os autores elaboram uma heurística para ser usada em instâncias maiores do problema: um algoritmo guloso que associa novas requisições ao conjunto de réplicas atual. Caso um aumento de requisições possa ser atendido pelo conjunto, o algoritmo procura uma réplica para remover. Caso contrário, o algoritmo

procura um servidor para inserir nova réplica. Os resultados obtidos com a heurística são muito próximos dos ótimos, com baixa replicação média, baixa distância média usuário-réplica e baixa quantidade de demandas não atendidas. Apesar de não tratar explicitamente da QoS, a abordagem limita a distância entre cliente e servidor que o atenderá em, no máximo, *d*; como a distância é considerada no trabalho uma medida genérica, não permitir que servidores "mais distantes do que *d*" atendam determinada requisição é comparável a uma exigência de qualidade. Na prática, *d* é tratado como uma restrição de QoS.

BEKTAS *et al.* (2007) abordam os problemas de Dimensionamento e Posicionamento de Servidores (PLS) e o PPRDR' estáticos. O PPRDR' se assemelha ao PPRDR (Problema de Posicionamento de Réplicas e de Distribuição de Requisições), mas não trata a garantia da QoS. Os autores propõem um modelo matemático não linear e uma linearização correspondente; um algoritmo exato; e um algoritmo heurístico guloso que ativa um servidor a cada iteração, segundo uma prioridade predefinida que pode ser, por exemplo, a ordem crescente do custo de ativação dos servidores. A resolução dos PPR e PDR considera os servidores ativos na RDC. O PDR é resolvido associando cada cliente ao servidor que gera o menor custo de comunicação. O PPR é resolvido calculando as economias de conteúdos, que representam o benefício de replicar um conteúdo em um servidor. Após calculado o benefício para cada conteúdo, os conteúdos são posicionados no servidor de acordo com uma ordem não crescente dos benefícios, sem violar as restrições de capacidade de armazenamento de cada servidor. Os resultados computacionais dos algoritmos propostos indicam bom desempenho, atingindo resultados distantes no máximo 5% em relação à solução ótima.

Em qualquer processo de acesso a dados, existe um tempo de latência ou espera entre a solicitação e a disponibilização dos dados – em redes, isso significa o tempo de trânsito de um "pacote" de dados entre os nós. HUANG e ABDELZAHER (2004) apresentam um mecanismo para estabelecer um limite de latência no acesso a conteúdos, podendo o limite ser especificado por classe de conteúdo, o que influencia diretamente a QoS. No trabalho proposto, utilizam uma modelagem por grafos, na qual os vértices representam os servidores escolhidos para o posicionamento das réplicas e cujo objetivo é construir um grafo dominante mínimo (grafos dominantes são grafos orientados nos quais cada par de vértices está ligado por uma e somente uma aresta – ou de ida, ou de volta, mas não ambas (BORTOLIN e GIRARDI, 2003)). O experimento

computacional compara o algoritmo proposto a outros algoritmos de dominação de grafos, e demonstra que a abordagem distribuída consegue encontrar soluções com um custo de replicação muito baixo, ao mesmo tempo respeitando os parâmetros de QoS estabelecidos. A comparação contra outros algoritmos de minimização de latência média mostra redução de quatro a cinco vezes na quantidade de violações de tempo de resposta. Uma grande vantagem da proposta é a flexibilidade de adaptação do sistema às mudanças nas condições da rede e dos servidores. Entre as principais limitações estão a necessidade de estabelecer mecanismos para que o atendimento aos parâmetros de QoS se faça com o mínimo de replicação possível e o fato de que o tamanho dos arquivos afeta os resultados, uma vez que cada tamanho tem uma latência e os arquivos solicitados podem ter qualquer tamanho. Outro estudo experimental mostra que a latência média no *download* de um arquivo é aproximadamente proporcional ao seu tamanho, se o arquivo tem entre 1 e 100 KB de tamanho. O trabalho, de 2004, considera o tamanho típico de arquivos a transmitir, coincidentemente, entre 1 e 100 KB, baseado em literatura existente à época (referências de 1996 e 2002, respectivamente CROVELLA e BESTRAVOS, *Self-similarity in WWW traffic: Evidence and possible causes* e SAROIU *et al.*, *An analysis of Internet content delivery systems*, citadas pelos autores). HUANG e ABDELZAHER (2004) reconhecem em uma seção no trabalho que trata de efeitos dos tamanhos de arquivo, que "para arquivos grandes (com multi-megabytes de tamanho, como clipes multimídia), a métrica de desempenho mais importante é a manutenção de uma taxa relativamente constante de *download*, mas esse não é o foco do presente sistema" (em tradução livre). Os arquivos que vêm sendo distribuídos por RDCs hoje, 10 a 15 anos depois, são muitas vezes multimídia e com tamanho da ordem de quatro mil vezes maior do que aqueles considerados pelos autores (vídeos típicos com cerca de 40 minutos de duração chegando a 450 MB), segundo NEVES (2011). Há, portanto, indicação de que a solução pode não ser apropriada para a realidade atual.

NEVES (2011) examina o PPRDR, uma variante do PPR, considerando o atendimento à garantia da Qualidade de Serviço (QoS) e a minimização do tráfego na rede. O PPRDR é composto pelos Problemas de Posicionamento de Réplicas (PPR) e de Distribuição de Requisições (PDR). O autor analisa e compara diversas abordagens exatas (através do uso de formulações matemáticas) e heurísticas (através de algoritmos construtivos e metaheurísticas), combinando-as para propor novos algoritmos

heurísticos e híbridos (exatos + heurísticos). O autor considera múltiplos conteúdos, de vários tamanhos, inclusive os extensos; a capacidade de armazenamento, a banda e a carga nos servidores; a banda mínima e máxima para os clientes; a possibilidade de uma requisição ser atendida por mais de um servidor ao mesmo tempo; e a possibilidade de uma requisição ser atendida em mais do que um período de tempo. Além disso, o autor considera que a localização dos servidores (PLS) é um dado do problema. Para o PPR, o autor propõe um algoritmo que combina o método ILSAM (*Iterated Local Search with Adaptive Memory*, que consiste da metaheurística ILS (*Iterated Local Search*) usando memória adaptativa (*Adaptive Memory*)) com a heurística ARTR (*Adaptive Record-To-Record*, que é a RTR com memória adaptativa). A ILSAM permite alterar os parâmetros do RTR (*Record-to-Record*) e a probabilidade de escolha das estratégias de perturbação entre diferentes estados: Normal, Intensificação e Diversificação, guardando na memória os resultados das iterações que servirão de base para a alternância entre estados. O algoritmo RTR percorre a vizinhança de uma solução semente de forma aleatória, podendo aceitar vizinhos piores do que a semente desde que o valor da função objetivo desse vizinho seja menor (para problemas de minimização) que o *Recorde* acrescido de um *Desvio*, onde *Recorde* é o valor da função objetivo da melhor solução até o momento, e *Desvio* é o valor que representa a distância máxima tolerada entre a melhor solução encontrada e uma outra solução a usar como nova semente. O valor de *Desvio* influencia diretamente o grau de diversificação do algoritmo: quanto maior, mais e diversas soluções serão usadas como semente e, portanto, maior a região do espaço de soluções que será explorada. Se *Desvio* for igual a zero, o algoritmo somente aceita soluções melhores do que a atual para nova semente (intensificação). A ARTR é uma variação da RTR que faz uso de memória adaptativa. A ARTR alterna entre três fases (Intensificação, Básico e Diversificação) através de mudanças no parâmetro *Desvio* do algoritmo RTR, em função da quantidade de iterações e do comportamento do processo (melhoras obtidas ou não). Além disso, vários algoritmos para estimar a demanda futura são analisados. Os resultados indicam que as heurísticas propostas geram soluções de boa qualidade, com redução de custo de até cinco ordens de grandeza comparadas à heurística usada em RDCs reais. No entanto, os resultados obtidos para o PPR em instâncias com mais do que 100 servidores não são conclusivos, devido ao reduzido número de iterações feitas pelos algoritmos.

SHAHABI e BANAEI-KASHANI (2002) pesquisam questões relacionadas à redução de custos totais (envolvendo comunicação e armazenamento) e à automação de algumas das tarefas de gerenciamento de uma RDC, desenvolvendo um mecanismo de entrega que envolve o posicionamento das réplicas ao longo dos servidores através de uma hierarquia de servidores com redundância.

WAUTERS *et al.* (2006) propõem um conjunto de algoritmos para posicionamento de réplicas (*Replica Placement Algorithms*, RPAs) baseados em uma formulação de programação linear inteira (PLI) do problema de posicionamento centralizado. Os algoritmos melhoram o desempenho de uma RDC otimizando a carga nos servidores e na rede, evitando congestionamentos e atrasos. O estudo trata RDCs com formato de anel, como apresentado na Figura 2, embora a proposta possa ser aplicada a diferentes topologias.

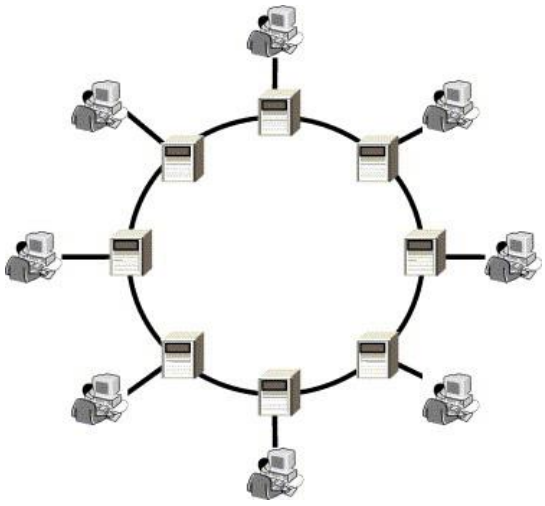

Figura 2. RDC em formato de anel, com conexões à rede, conforme WAUTERS *et al.* (2006).

Os autores acreditam que, com cenários estáticos, o acréscimo de capacidade de armazenamento seja capaz de prover boa relação custo-benefício e este cenário é por eles cogitado para exame em trabalhos futuros. O trabalho apresenta heurísticas que adaptam dinamicamente o posicionamento das réplicas às variações de comportamento dos usuários, disponibilidade de conteúdo e carga na rede. Os autores observam que estas estratégias se mostram adequadas para reduzir o congestionamento na rede e prover serviço mais confiável, ao preço de uma carga de rede ligeiramente maior. A representação de custos das conexões nos algoritmos de balanceamento de carga, como

atrasos na propagação que devem ser minimizados em conjunto com outros recursos, poderia, segundo os autores, ser uma área de estudo futura. Como o estudo se baseia no armazenamento de arquivos inteiros, método efetivo para serviços do tipo vídeo-sob-demanda, os autores sugerem também como trabalho futuro um método para armazenar arquivos parciais para conteúdos muito populares como transmissão ao vivo de TV.

ZHOU e XU (2007) investigam a replicação de conteúdo ao longo de um conjunto de servidores para aliviar as restrições de capacidade e de rede e prover alta qualidade e disponibilidade aos serviços de distribuição de conteúdo. Eles enunciam o problema como um problema de otimização combinatória com os objetivos de maximizar o número de réplicas de cada vídeo e balancear a carga dos servidores, considerando as limitações de capacidade e de largura de banda dos servidores. Abordam o PR, Problema de Replicação de conteúdos, propondo três algoritmos de replicação. O primeiro algoritmo cria réplicas priorizando os conteúdos de maior custo de comunicação. O segundo se baseia na popularidade do conteúdo pressupondo uma distribuição de popularidade. O terceiro segue uma estratégia *round-robin* (algoritmo de escalonamento em que cada processo numa fila circular é atendido durante certo período de tempo) com pesos, baseada na popularidade dos conteúdos, replicando mais vezes os mais populares. Os autores também abordam o PPR, propondo para sua solução dois algoritmos. No primeiro, afirmam que quando os algoritmos de replicação consideram o fato de que todas as réplicas têm um custo de comunicação uniforme, então um algoritmo *round-robin* resulta na solução ótima. Em casos reais, sabe-se que os custos de comunicação não são uniformes; por isso, os autores propõem o segundo algoritmo, que posiciona réplicas de conteúdo com grande custo de comunicação em servidores com menor carga, garantindo que os servidores não tenham mais do que uma réplica do mesmo conteúdo. Eles também propõem uma heurística baseada em *Simulated Annealing* para uma generalização do problema onde um mesmo conteúdo pode ter vários graus de qualidade. Os algoritmos são testados em um cenário composto para distribuição de vídeos, com servidores providos de grande capacidade de armazenamento, com grande largura de banda e conteúdos extensos (90 minutos). Os resultados obtidos mostram que os algoritmos propostos conseguem soluções de alta qualidade que, em alguns casos, eram as soluções ótimas. Uma das limitações do trabalho é que os algoritmos propostos consideram conhecidos os dados sobre a popularidade dos conteúdos, o que na prática não é usual.

2.2.2 A metaheurística POPMUSIC e aplicações

A metaheurística POPMUSIC (*Partial OPtimization Metaheuristic Under Special Intensification Conditions*, Metaheurística de Otimização Parcial sob Condições Especiais de Intensificação) é um método geral projetado para lidar com instâncias grandes de problemas de otimização combinatória difíceis, através de sua decomposição em problemas menores. O método pode ser visto como uma busca local trabalhando com vizinhanças grandes. De acordo com TAILLARD e VOSS (2001), não é fácil descobrir qual a maneira apropriada de decompor um problema, *a priori*. A decomposição de um problema em subproblemas pode levar a soluções de qualidade apenas moderada, já que os subproblemas podem ter sido criados de maneira arbitrária. Alternativamente, POPMUSIC parte de uma solução disponível para o problema (não necessariamente boa), e tenta melhorar a qualidade da solução de subconjuntos da solução incial, *a posteriori*. A solução inicial é decomposta em "partes"; uma parte é escolhida como "semente", a partir da qual forma um subproblema, de tamanho definido. Cada subproblema é composto por uma coleção das partes mais fortemente relacionadas à semente. Algum método de otimização (exato ou heurístico) é empregado para tentar melhorar localmente o subproblema. POPMUSIC organiza o processo de montagem e tratamento de cada subproblema, e de transposição das melhorias eventualmente obtidas durante o tratamento de cada subproblema para o problema completo, até que toda a solução tenha sido examinada e não exista mais nenhum subproblema que possa ser melhorado. Se a definição das partes e subproblemas são feitas de maneira adequada, a cada melhora obtida para o subproblema corresponde uma melhora no problema completo.

O pseudocódigo na Figura 3, seguido por uma breve explicação, ilustra um esquema geral para o método POPMUSIC.

procedimento POPMUSIC (S, r)

1. Dados a solução inicial S, composta pelas partes s_1, ..., s_p ; e r, tamanho dos subproblemas
2. $O = \{s_1, ..., s_p\}$ /* conjunto de controle
3. **Enquanto** $O \neq \emptyset$ **faça**
4. Selecione uma parte $s_j \in O$ /* parte semente

5. Monte subproblema R_j composto de r partes de S, incluindo s_j e as r -1 partes mais fortemente relacionadas a s_j
6. Submeta R_j ao processo de tentativa de melhoria
7. **Se** R_j tiver sido melhorado
8. Atualize na solução corrente S as r partes do subproblema otimizado R_j
9. Inclua em O todas as partes do subproblema otimizado R_j que não estejam lá
10. **Senão**
11. Remova de O a parte semente s_j
12. **fim Enquanto**

fim POPMUSIC.

Figura 3. Pseudocódigo de POPMUSIC baseado em TAILLARD e VOSS (2001).

Sejam S uma solução de um determinado problema que se deseja melhorar e r, único parâmetro do método, o tamanho dos subproblemas. Suponha que S possa ser definida como um conjunto de p partes s_1, ..., s_p (linha 1), e que uma medida de relacionamento possa ser definida entre quaisquer duas partes. Para evitar gerar o mesmo subproblema mais do que uma vez, implementa-se um procedimento de controle: um conjunto O armazena as partes que podem ser usadas como sementes para definir um subproblema que possa ser tratado em busca de melhora. O conjunto O é inicializado na linha 2. Na linha 4, POPMUSIC seleciona uma parte s_j, à qual chama parte semente. Na linha 5, são selecionadas r-1 < p partes mais fortemente relacionadas a s_j para formar um subproblema R_j de tamanho r. O subproblema é submetido a uma tentativa de melhoria (linha 6). Quando O estiver vazio, então todos os subproblemas terão sido examinados, não restando algum que possa ser melhorado, e o processo acaba (linhas 3 e 12). Durante a aplicação do algoritmo, o subproblema R_j pode ter sido melhorado (linha 7). Nesse caso, algumas de suas partes terão sido alteradas e a solução S é atualizada (linha 8). Com a alteração das partes, novas melhorias podem ser possíveis em subproblemas criados a partir daquelas partes; logo, tais subproblemas devem ser reexaminados e, portanto, as partes que eventualmente tenham sido retiradas anteriormente são reinseridas no conjunto O (linha 9). Caso não tenha sido possível melhorar o subproblema, a parte semente correspondente é retirada de O (linha 11).

ALVIM e TAILLARD (2013) lembram que a complexidade do método pode ser muito alta, uma vez que *O* não é obrigatoriamente reduzido a cada iteração. Entretanto, diversas implementações mostram de forma empírica que o número de iterações de um algoritmo baseado neste procedimento cresce quase linearmente com *p*. Nos experimentos relatados no Capítulo 4, verificou-se que a quantidade média de iterações foi sempre em torno de duas vezes o tamanho da instância (a média das quantidades de iteração / tamanho da instância foi de 2,18).

Além do tamanho dos subproblemas, o parâmetro *r*, a metaheurística deixa alguns componentes livres, para que sejam especificados de acordo com o problema a resolver:

- o procedimento para obter uma "solução inicial";

- a definição do que é uma "parte" da solução;

- a "função de relacionamento" entre as partes para criar um subproblema;

- o procedimento de seleção de uma parte ("semente") que será usada para dar origem a um subproblema;

- o procedimento para "otimizar" os subproblemas.

Este método tem se mostrado altamente eficiente em resolver problemas grandes, conforme demonstrado, por exemplo, nos trabalhos de ALVIM e TAILLARD (2009), ALVIM e TAILLARD (2013) e OSTERTAG *et al.* (2009).

A seguir, uma breve revisão sobre trabalhos baseados no método POPMUSIC.

Segundo SUBRAMANIAN (2012), há um grande interesse no estudo do Problema de Roteamento de Veículos (em inglês, *Vehicle Routing Problem* – VRP), devido tanto à sua importância prática quanto à dificuldade de resolvê-lo. O VRP consiste basicamente em definir as melhores rotas para que veículos possam servir a clientes, respeitadas algumas restrições. TAILLARD e VOSS (2001) aplicam em seu trabalho o *framework* POPMUSIC ao VRP, com o objetivo de definir um conjunto de viagens de veículos "saindo de" e "retornando a" um depósito, de forma que as viagens passem por todos os clientes, sendo cada cliente visitado uma única vez; a soma das demandas dos clientes em uma viagem não pode exceder a capacidade do(s) veículo(s) que os atendem. Para estabelecer o que será considerada uma parte da solução, estabelecem duas opções: na primeira, uma parte é a viagem de um veículo e, a solução,

um conjunto de viagens; na segunda, uma parte é o conjunto de percursos "chegando a" e "partindo de" um cliente e a solução, um conjunto de percursos, indo de um cliente a outro, desse a um terceiro e assim por diante. Na segunda opção, a função de relacionamento entre duas partes é definida de forma dependente da função objetivo a otimizar. Se o objetivo for minimizar a distância total percorrida pelos veículos, a relação é o inverso da distância entre dois clientes. Se o objetivo for minimizar a quantidade de veículos em serviço (com o objetivo secundário de minimizar a distância total percorrida), é aplicada uma penalidade à distância original entre os dois clientes caso eles estejam em percursos diferentes. A escolha do cliente semente para gerar cada subproblema é feita de maneira aleatória. É possível citar outros trabalhos que aplicam POPMUSIC ao VRP, como por exemplo os de OSTERTAG *et al.* (2009) e de LI *et al.* (2007). Este último aborda o HVRP (VRP Heterogêneo, quando a frota é composta por diferentes tipos de veículo, com variações de capacidade, custo fixo e custo variável em função da distância). Os autores afirmam que as heurísticas foram capazes de produzir soluções próximas do ótimo, com baixo esforço computacional para o problema abordado.

Outro exemplo de aplicação do POPMUSIC é para o problema de balanceamento de partes mecânicas, necessário, entre outros casos, para o funcionamento equilibrado do rotor de uma turbina. O rotor é um eixo onde são fixadas lâminas que deveriam ter peso exatamente igual mas que, na prática, têm pesos diferentes. As diferenças de pesos têm que ser equilibradas para que o centro de massa do conjunto se aproxime de – idealmente coincida com – o centro do eixo da turbina, para minimizar as vibrações que podem até destruir a turbina. TAILLARD e VOSS (2001) aplicam POPMUSIC ao problema de minimizar a distância do centro de massa do conjunto ao centro do eixo, usando como procedimento de melhoria uma heurística baseada em Busca Tabu (GLOVER, 1986). A solução inicial é escolhida de maneira aleatória. Uma parte é uma lâmina. A escolha da lâmina semente é realizada considerando-se as lâminas em ordem não-crescente de peso. O método que usa POPMUSIC com Busca Tabu se mostrou mais eficiente do que a aplicação direta da Busca Tabu para instâncias grandes do problema.

ALVIM e TAILLARD (2009) propõem uma aplicação da metaheurística POPMUSIC para o problema de rotulação cartográfica onde deseja-se posicionar rótulos em pontos de um mapa de forma a evitar sobreposição entre eles melhorando a

legibilidade. Experimentos computacionais demonstram que o método obteve melhores resultados do que outros métodos relatados na literatura; testes com instâncias envolvendo até 10 milhões de pontos mostram que o tempo computacional cresce quase linearmente com o tamanho do problema.

Uma das tarefas mais complexas do método POPMUSIC, em termos algorítmicos, é a obtenção de uma solução inicial. ALVIM e TAILLARD (2013) apresentam uma forma de gerar uma solução inicial apropriada para instâncias do LRP (*Location-routing Problem*, Problema de Localização e Roteamento) com milhões de clientes, usando POPMUSIC. Resolver o LRP envolve simultaneamente definir a localização de depósitos entre possíveis opções e estabelecer rotas de entrega para um conjunto de clientes de forma a minimizar o custo total da solução. Os autores definem os componentes principais do método POPMUSIC: uma parte como um percurso de um veículo; o relacionamento entre partes como a distância mínima entre dois clientes pertencentes a diferentes percursos; e o procedimento de melhoria como um método baseado em Busca Tabu (GLOVER, 1986). Experimentos computacionais mostram que o método POPMUSIC pode ser aplicado a instâncias do LRP de tamanhos diversas ordens de magnitude superiores aos das instâncias comumente tratadas na literatura.

A heurística proposta para o PPRDR, apresentada no próximo capítulo, usa o método POPMUSIC.

3. POPMUSIC aplicado ao Problema de Posicionamento de Réplicas e Distribuição de Requisições em RDCs

No capítulo anterior apresentou-se o *framework* POPMUSIC – *Partial OPtimization Metaheuristic Under Special Intensification Conditions* – metaheurística especialmente útil para criar métodos heurísticos que tratam de instâncias grandes de problemas combinatórios difíceis. Este capítulo detalha o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR) e explicita as definições adotadas para a aplicação de POPMUSIC ao PPRDR em instâncias de RDCs de médio e grande portes.

3.1 O Problema de Posicionamento de Réplicas e Distribuição de Conteúdos (PPRDR)

De uma forma geral, RDCs são empreendimentos comerciais, que visam lucro e nesses casos é desejável que seus custos sejam mantidos no menor nível possível. No entanto, muitos clientes, em especial os corporativos, contratam serviços de *download* de conteúdo com exigências de nível mínimo de atendimento (*Quality of Service* – QoS). As requisições por conteúdos não são completamente previsíveis. Para garantir o nível de atendimento desejado pelos clientes, o ideal seria dispor de réplica de todos os conteúdos em todos os servidores da RDC. Entretanto, diversos tipos de restrições como capacidade de armazenamento, custo e capacidade de transmissão (“banda”) de cada servidor e, eventualmente, até restrições contratuais sobre a quantidade total de réplicas de conteúdo que podem ser feitas, impedem ou ao menos desaconselham semelhante esquema de replicação. Dessa forma, em um gerenciamento de RDC, considerando-se as requisições dos clientes, torna-se necessário escolher quais as réplicas de conteúdos armazenar em cada servidor. As várias alternativas de posicionamento de réplicas e de

recebimento de requisições compõem um problema combinatório conhecido na literatura (BEKTAS *et al.*, 2007, NEVES, 2011) como o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR), uma variante do PPR (Problema de Posicionamento de Réplicas) que considera sua solução em conjunto com a solução do PDR (Problema de Distribuição de Requisições).

O trabalho de NEVES (2011) trata vários problemas, entre eles o PPRDR. A seguir, a definição do PPR, PDR e do PPRDR segundo o autor:

- dados um conjunto de servidores e um conjunto de conteúdos, o PPR consiste em definir em quais servidores posicionar réplicas de quais conteúdos a fim de reduzir os custos de transmissão;

- dado um conjunto de servidores com réplicas posicionadas, o PDR consiste em encontrar a melhor distribuição das requisições para os servidores a fim de reduzir os custos de transmissão;

- por fim, o PPRDR consiste em encontrar o melhor posicionamento para as réplicas e distribuir as requisições entre os servidores, com o objetivo de reduzir o tráfego na RDC respeitando as restrições de QoS das requisições.

O PPRDR difere do PPR por tratar as requisições de forma individual e não aglomerada; uma requisição pode ser tratada por mais de um servidor, se isso for vantajoso. Uma solução para o PPRDR é uma associação de conteúdos a servidores, respeitando a restrição de capacidade do servidor, e uma associação de requisições a servidores, respeitando a restrição de existir o conteúdo requisitado nos servidores.

Há duas versões do PPRDR: *on-line* e *off-line*. A versão *on-line* considera a natureza dinâmica do problema (surgimento de requisições, conteúdos etc.), enquanto a versão *off-line* considera rede, conteúdos e demandas conhecidos *a priori* e é utilizada para testes, estabelecimento de limites e nos casos onde os dados do problema apresentam pouca ou nenhuma variação. O presente trabalho trata a versão *off-line*, para a qual NEVES (2011) apresenta, entre outras possíveis soluções, formulações matemáticas que incluem diversas características reais do problema (a mais abrangente delas está reproduzida ao final deste capítulo), e duas heurísticas baseadas na metaheurística ILS (uma delas, a ILSAM, com memória adaptativa, é usada neste estudo).

Ainda no trabalho de NEVES (2011), para resolver de forma heurística o PPRDR, o autor propõe o algoritmo HNH (*Hybrid Network Heuristic*), em português Heurística de Rede Híbrida. Experimentos conduzidos no trabalho mostram excelentes resultados, em termos de qualidade da solução e de tempos computacionais para a versão *on-line* do problema, quando comparadas às outras diversas heurísticas testadas, incluindo GHS (*Global Hosting System*), solução patenteada de um provedor de RDC, OGHS (*Optimized Global Hosting System*), HC (*Hybrid Constructive*) e HCAGAP (*Hybrid Constructive Adapted Generalized Assignment Problem*). HCAGAP, que fornece uma solução ótima para o PPR, dada uma certa demanda, usa a formulação AGAP (*Adapted Generalized Assignment Problem*), uma variante do Problema de Alocação Generalizado, e a mesma formulação é empregada pela HNH, contextualizando o PPR como uma variante do Problema da Mochila (MARTELLO e TOTH, 1990). Para prever demandas futuras, HNH usa vários estimadores, incluindo (para a versão *off-line*) o estimador "conhecimento futuro", que usa o conhecimento da demanda real do próximo período, conforme definido em NEVES (2011). Embora na vida real não seja possível conhecer a demanda futura com precisão, o conhecimento futuro na versão *off-line* permite estabelecer limites para outras abordagens e medir a precisão de outros estimadores. A resolução do PDR pela HNH é dada pela solução do Problema de Fluxo de Custo Mínimo (PFCM) em redes construídas pelo modelo de fluxo, usando o algoritmo Network Simplex (AHUJA *et al.*, 1993) disponível no resolvedor CPLEX, capaz de fornecer solução exata em um tempo computacional menor do que o tempo obtido por outros métodos. HNH pode ser descrita pelo algoritmo a seguir.

procedimento HNH

1. **Para todo** período de tempo **faça**
2. Resolva PDR: modelo de Fluxo em Rede / Network Simplex
3. Use Estimador: Conhecimento Futuro
4. Resolva PPR: formulação AGAP
5. **fim Para todo**

fim HNH.

Figura 4. Algoritmo de *Hybrid Network Heuristic* (HNH) para versão *off-line* do PPRDR, baseado em NEVES (2011).

O modelo do PPRDR em RDCs utilizado por NEVES *et al.* (2008) e NEVES (2011) usa três entidades: Servidor, Réplica e Requisição. A entidade "Servidor" representa os servidores na rede. "Réplica" representa uma cópia de um conteúdo. Um mesmo conteúdo pode estar replicado em mais de um servidor. "Requisição" representa a demanda de um cliente por conteúdo. Uma solução, nesse modelo, é uma associação entre servidores, réplicas e requisições, indicando quais réplicas armazenar em cada servidor e quais requisições (ou parcelas de requisições) atender a partir de cada servidor, em cada período de tempo considerado. Para atender a uma dada requisição, um servidor tem que conter uma réplica do conteúdo desejado; caso isso não ocorra, a viabilidade da solução pode ser alcançada pela redistribuição da requisição a outro servidor. Todas as requisições devem ser atendidas. Cada requisição pode ser atendida por um ou mais servidores. Nem todos os servidores precisam ser utilizados em certo período.

Para maior similaridade com casos do mundo real, NEVES (2011) discretizou o horizonte de planejamento em períodos de tempo *t*, ou simplesmente períodos, e considerou conteúdos grandes que não são inteiramente transmissíveis em um único período.

São dados do PPRDR *off-line*:

- o conjunto C de réplicas de conteúdos, sendo que cada réplica k, para $k = 1, 2, ..., |C|$, traz associada seu tamanho L_k;

- o conjunto R de requisições ("requisição" e "cliente" são conceitos similares para efeitos deste trabalho), sendo que cada requisição i, para $i = 1, 2, ..., |R|$, traz associada o atraso máximo permitido TD_i, e a banda mínima requerida BR_i por período. O atraso máximo e a banda mínima são exigências de acordo com o nível de QoS. Outras informações que vêm com a requisição i são: a sua banda máxima BX_i, que pode influenciar na qualidade da entrega, permitindo reduzir o tempo gasto na transmissão para um nível melhor do que o mínimo exigido, caso haja disponibilidade por parte do servidor; o servidor ao qual a requisição i foi originalmente dirigida ($origem(i)$); e o atraso local ($ld(i)$) referente ao tempo de transmissão de um pacote de dados entre o cliente demandante da requisição e o servidor ao qual está conectado, ($origem(i)$);

- o conjunto V de servidores. Associado a cada servidor j, para $j = 1, 2, ..., |V|$, são dados: a distância de j em relação a cada um dos outros servidores l, para $l = 1, 2, ..., |V|$ e $l \neq j$, pertencentes a V (dada pelo RTT (*Round Trip Time*) médio (tempo de transferência de dados do servidor j a outro l e deste de volta a j) ao longo de todo o horizonte de planejamento), sua capacidade máxima de armazenamento (AS_j) e sua capacidade máxima de transmissão (MB_j), bem como o conjunto de réplicas nele armazenadas, C_j. Com a definição do atraso local ($ld(i)$) entre o cliente demandante da requisição e o servidor de origem, e o atraso entre qualquer par de servidores j e l em V, ($atraso(j,l)$), pode-se calcular o atraso para qualquer requisição.

Para manter a semelhança com casos reais, a duração total de cada rodada de testes foi definida com um tempo limitado: estabeleceu-se o prazo de duas horas como critério de parada. NEVES (2011) teve sucesso na solução do PPRDR *off-line* com a aplicação da metaheurística ILS (*Iterated Local Search*) e heurística RTR (*Record-To-Record*) usando memória adaptativa – respectivamente ILSAM e ARTR – para instâncias com até 100 servidores. Entretanto, conforme observado pelos autores, o tempo médio por iteração cresce de modo não-linear em relação ao tamanho das instâncias, e não se conseguiu obter resultados conclusivos nos testes aplicados a instâncias com maior quantidade de servidores pelo reduzido número de iterações havido.

3.2 POPMUSIC aplicada ao PPRDR

O presente trabalho emprega a metaheurística POPMUSIC, proposta por TAILLARD e VOSS (2001), apresentada no capítulo anterior, para propor o Algoritmo POP-PPRDR.

O único parâmetro do método é r, a quantidade de partes em um subproblema. O valor de r foi escolhido experimentalmente conforme descrito na Seção 4.3. A seguir, apresentam-se as escolhas dos componentes livres de POPMUSIC, empregadas para a aplicação do método à resolução do PPRDR *off-line*:

- Solução inicial: Foi selecionada a heurística HNH proposta por NEVES (2011) para geração da solução inicial de POPMUSIC quando da resolução do PPRDR *off-line*, pelos bons resultados apresentados na resolução da versão *on-line* do problema, tanto em termos de tempos computacionais quanto em termos da qualidade das soluções obtidas. Na versão *off-line* os dados de entrada são conhecidos *a priori*, e

assim o uso do estimador “Conhecimento Futuro” na fase de previsão de demanda da heurística HNH é possível. Uma solução é composta pela associação de cada um dos servidores utilizados – pode haver servidores da instância sem utilização – com suas respectivas réplicas de conteúdo e uma parcela (entre 0 a 100%) de atendimento a requisições para respectivos conteúdos. Existe a possibilidade de uma requisição ser atendida por mais de um servidor.

- Parte da solução: Uma parte s_j é definida como sendo composta por um servidor j com seus respectivos conteúdos C_{jt} nos períodos de tempo t, para t = 1, 2, ..., $|T|$, e os percentuais de cada uma das requisições que serão atendidas por j (Q_{jt}) nos períodos t, para t = 1, 2, ..., $|T|$. Observando a definição, fica claro que há uma relação biunívoca entre cada servidor e cada parte. A definição para uma parte s_j (servidor + conteúdo + parcela de requisição atendida pelo servidor), ao prever atendimento de parcelas de uma requisição e distribuir cada parcela a um único servidor, torna as partes disjuntas, garantindo que não haja interferência entre elas; em cada período, há somente um servidor atendendo cada parcela específica de uma requisição. Por extensão, não ocorrem interferências entre subproblemas, o que permite dividir a solução inicial, que forma o problema original, em subproblemas independentes entre si. Na Figura 5, a seguir, estão representadas e identificadas algumas das partes em uma solução inicial hipotética.

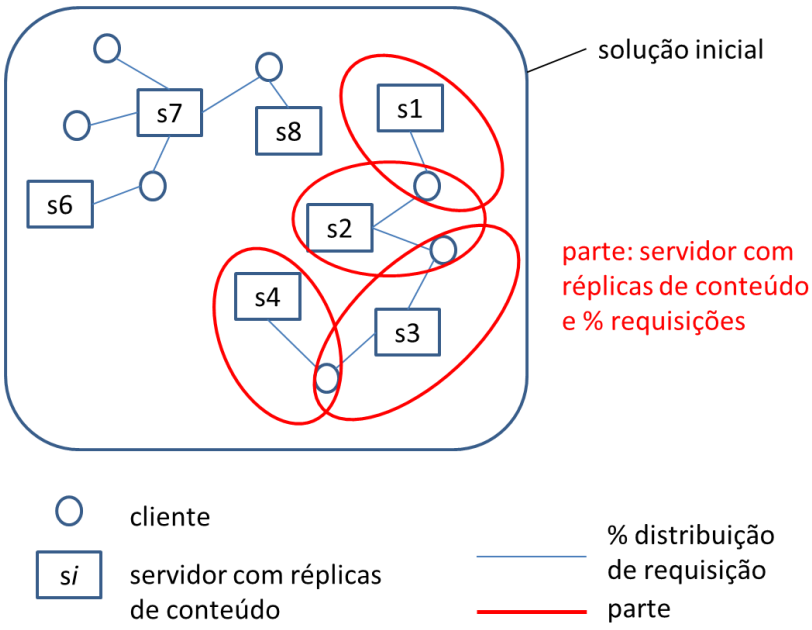

Figura 5. Representação de solução inicial identificando algumas partes.

- Relação entre duas partes: A relação entre duas partes foi definida em função da distância entre seus respectivos servidores. O relacionamento entre duas partes é dado pelo inverso da distância entre seus respectivos servidores. A distância entre cada par de servidores é um dado do problema, expressa pelo RTT médio entre tal par de servidores ao longo de todo o horizonte de planejamento.

De acordo com NEVES *et al.* (2008), a decisão sobre o posicionamento das réplicas no PPRDR depende, além da capacidade de armazenamento dos servidores, também da distância entre eles. Ainda de acordo com os autores, cenários típicos contêm uma quantidade muito maior de clientes do que servidores em uma RDC. Para simplificação do modelo, decidiu-se considerar conectados a cada servidor seus vários respectivos clientes e concentrar os esforços de melhoria de resultados na replicação de conteúdos e na distribuição de requisições entre os servidores. O subproblema R_j é montado com a parte semente s_j e as r-1 partes mais fortemente relacionadas (mais próximas) de s_j. Pode-se observar na Figura 6 a representação de um subproblema de tamanho 3, montado a partir da solução inicial.

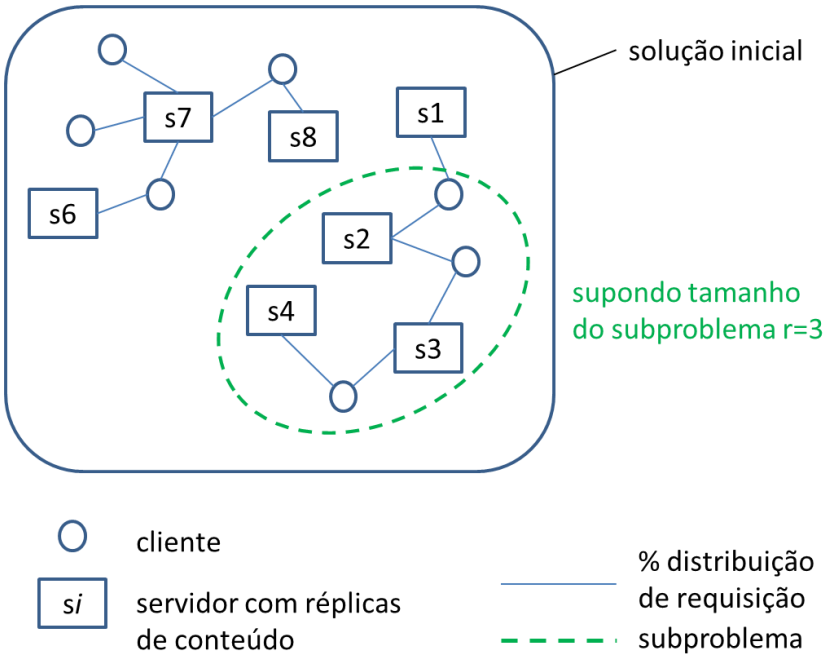

Figura 6. Representação de subproblema composto por três partes da solução inicial.

- Seleção de parte semente: As partes disponíveis para servir de semente ficam organizadas num conjunto *O* (que inicialmente contém todas as partes da solução inicial), tratado como uma fila seguindo a estratégia FIFO (*First In, First Out*). A

cada iteração, a parte no início da fila serve de semente para montar um subproblema que é então examinado. Caso não haja melhoria, a semente é retirada de *O*. Caso haja, todas as partes do subproblema que não estejam em *O* são ali reinseridas, depois do último elemento. Quando *O* estiver vazio, o algoritmo para.

- Procedimento de melhoria para otimizar subproblemas: Cada subproblema criado é submetido ao algoritmo de melhoria proposto por NEVES (2011) para solução do modelo *off-line* do PPRDR, empregando ILSAM com ARTR. Os algoritmos ILSAM e ARTR foram brevemente descritos na Seção 2.2.1; mais detalhes podem ser encontrados no trabalho de NEVES (2011). O ILSAM começa sempre no estado N, Normal. A mudança entre estados é definida por um parâmetro, max_iterações, o número máximo de iterações sem melhora permitido em cada etapa. Ao atingir max_iterações em N, o algoritmo verifica há quantas iterações a melhor solução corrente foi encontrada. Se a quantidade for menor do que max_iterações, a busca deve continuar na vizinhança próxima; o algoritmo passa para o estado I, Intensificação. A mesma verificação é feita a cada max_iterações, mas em I, se a quantidade de iterações desde a última melhor solução corrente for menor do que o parâmetro, prossegue-se em I; se maior, alterna-se para N. Em N, novas iterações acontecem e outro teste é realizado quando o limite do parâmetro é atingido. Se a melhor solução corrente estiver distante (encontrada há mais do que max_iterações), a busca deve procurar soluções em regiões mais afastadas, passando a uma etapa de Diversificação, D1. Em D1, a verificação após max_iterações pode indicar melhora próxima, caso em que o algoritmo volta ao estado N, ou distante, caso em que passa-se ao estado D2. Finalmente, em D2, a quantidade de max_iterações indica retorno ao estado N. NEVES (2011) determinou experimentalmente o valor de 10 para max_iterações. O funcionamento do ILSAM está representado a seguir.

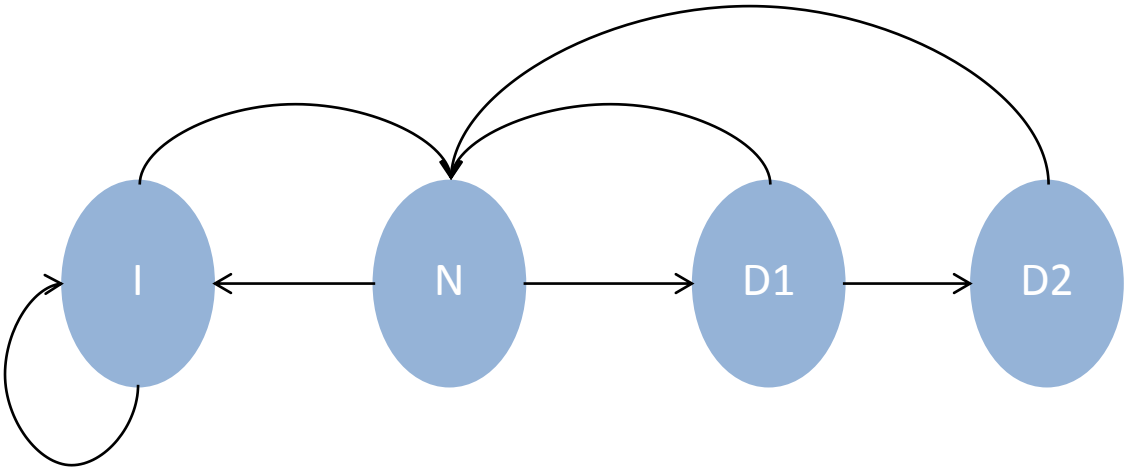

Figura 7. Diagrama de Estados do ILSAM segundo NEVES (2011).

Diferentemente do que ocorre no trabalho de NEVES (2011), onde o critério de parada é o tempo, neste trabalho o critério de parada utilizado para o ILSAM é baseado na ocorrência de certa quantidade de iterações sem melhora. Mais detalhes podem ser encontrados na Seção 4.2.

Para uma solução do PPRDR, considerou-se conforme NEVES (2011):

- Múltiplo atendimento: atendimento de uma mesma requisição por múltiplos servidores, cada servidor atendendo a uma fração do conteúdo demandado;
- Múltiplos conteúdos sendo requisitados e distribuídos no mesmo período;
- Atendimento maximizado: se a banda máxima do servidor (MB_j) e a banda máxima oferecida pelo cliente requisitante (Bx_i) são maiores do que a banda mínima exigida pela requisição (BR_i) e estão disponíveis, usa-se a disponibilidade para acelerar a entrega do conteúdo solicitado além do mínimo exigido;
- Múltiplas origens para os conteúdos.

Considerou-se também a função objetivo chamada Formulação Dinâmica (FD) no trabalho de NEVES (2011) para avaliar a qualidade de uma dada solução. Esta função considera todos os aspectos abordados no trabalho, incluindo múltiplas origens e atualização dos conteúdos (levando em conta o tempo de vida dos mesmos, período em que ficam disponíveis na RDC).

A formulação da função objetivo e suas restrições associadas são apresentadas nas equações numeradas de 01 a 17, após a lista de variáveis e constantes que se segue.

Variáveis:

- x_{ijt} fração do conteúdo solicitado pela requisição *i* entregue pelo servidor *j* no período *t*
- y_{kjuv} indicador de presença de réplica do conteúdo *k* no servidor *j* nos períodos de *u* até *v*; vale 1 se o conteúdo está, 0 em caso contrário
- w_{kjlt} indicador de cópia do conteúdo *k* pelo servidor *j* a partir do servidor *l* no período *t*; vale 1 se o conteúdo é copiado, 0 em caso contrário
- b_{it} *backlog* da requisição *i* no período *t*

Constantes:

- *Q* conjunto de requisições a atender.
- *V* conjunto de servidores da RDC.
- *C* conjunto de conteúdos.
- *t* período de tempo.
- δ duração do período de tempo, definida como 60 segundos.
- *T* conjunto de períodos de tempo.
- *k* réplica de conteúdo, cada réplica com seu tamanho L_k.
- B_k período em que o conteúdo *k* é disponibilizado na RDC.
- E_k período em que o conteúdo *k* é retirado da RDC.
- O_k servidor origem do conteúdo *k*.
- AS_j espaço em disco no servidor *j* (em bytes).
- MB_j banda máxima (capacidade de transmissão) do servidor *j* (em bytes/segundo).
- D_{it} demanda da requisição *i* no período *t* (em bytes).
- BR_i banda mínima exigida pela requisição *i* (em bytes/segundo). Indicada no acordo de nível de serviço (QoS).
- BX_i banda máxima aceita pela requisição *i* (em bytes/segundo).
- $g(i)$ conteúdo exigido pela requisição *i*.
- c_{ijt} custo de atendimento da requisição *i* pelo servidor *j* no período *t*, calculado como $(RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i)) \times BR_i$ se $(atraso_{j,origem(i),t} + ld(i))$

for menor ou igual do que TD_i. Caso contrário, o cálculo é feito como $(RTT_{j,origem(i),t} + atraso_{j,origem(i),t} + ld(i))$ x BR_i + 1000 x $(atraso_{j,origem(i),t} + ld(i) - TD_i)$ + 1000. $RTT_{j,l}$ (*RoundTrip Time*) é definido como ($atraso(j,l)$ + $atraso(l,j)$), o tempo para um pacote de dados ir de um servidor j a outro servidor l e voltar ao primeiro.

- p_{it} penalidade por usar *backlog* da requisição i no período t, dada por max(c_{ijt}) x 2, para todo $i \in Q$, para todo $t \in T$.

- h_{kjlt} custo de replicar conteúdo k no servidor j a partir do servidor l no período t. Nos experimentos efetuados por NEVES (2011), esse custo é sempre dado pelo tamanho do conteúdo k.

Formulação e restrições:

$$Min \sum_{i \in Q} \sum_{j \in V} \sum_{t \in T} c_{ijt} x_{ijt} + \sum_{i \in Q} \sum_{t \in T} p_{it} b_{it} + \sum_{k \in C} \sum_{j \in V} \sum_{l \in V} \sum_{t \in T} h_{kjlt} w_{kjlt} \quad (01)$$

S.a.

$$\sum_{j \in V} L_{g(i)} x_{ijt} - b_{i(t-1)} + b_{it} = D_{it} \quad \forall i \in Q, \forall t \in \left[B_{g(i)}, E_{g(i)}\right], \quad (02)$$

$$\sum_{i \in Q} L_{g(i)} x_{ijt} \leq \delta MB_j \quad \forall j \in V, \forall t \in T, \quad (03)$$

$$\sum_{j \in V} L_{g(i)} x_{ijt} \leq \delta BX_i \quad \forall i \in Q, \forall t \in T, \quad (04)$$

$$\sum_{j \in V} \sum_{t \in T} x_{ijt} = 1 \quad \forall i \in Q, \quad (05)$$

$$y_{g(i)jt} \geq x_{ijt} \quad \forall i \in Q, \forall j \in V, \forall t \in T, \quad (06)$$

$$\sum_{j \in V} y_{kjt} \geq 1 \quad \forall k \in C, \forall t \in \left[B_k, E_k\right], \quad (07)$$

$$y_{kjt} = 0 \quad \forall k \in C, \forall j \in V, \forall t \notin \left[B_k, E_k\right], \quad (08)$$

$$y_{kO_kB_k} = 1 \quad \forall k \in C, \quad (09)$$

$$y_{kjB_k} = 0 \quad \forall k \in C, \forall j \in \left\{V \mid j \neq O_k\right\}, \quad (10)$$

$$y_{kj(t+1)} \leq \sum_{l \in V} w_{kjlt} \quad \forall k \in C, \forall j \in V, \forall t \in T, \tag{11}$$

$$y_{kjt} \geq w_{kljt} \quad \forall k \in C, \forall j,l \in V, \forall t \in T, \tag{12}$$

$$\sum_{k \in C} L_k y_{kjt} \leq AS_j \quad \forall j \in V, \forall t \in T, \tag{13}$$

$$x_{ijt} \in [0,1] \quad \forall i \in Q, \forall j \in V, \forall t \in T, \tag{14}$$

$$y_{kjt} \in \{0,1\} \quad \forall k \in C, \forall j \in V, \forall t \in T, \tag{15}$$

$$b_{it} \geq 0 \quad \forall i \in Q, \forall t \in T, \tag{16}$$

$$w_{kjlt} \in \{0,1\} \quad \forall j,l \in V, \forall k \in C, \forall t \in T \tag{17}$$

Para reduzir o atendimento de requisições por servidores muito distantes, privilegiando a escolha de servidores que possam atender as requisições dentro do atraso máximo estipulado, a formulação considera tais fatores em c_{ijt} (o custo de entrega da fração dos conteúdos solicitados *i* pelo servidor *j* no período *t*) e procura minimizá-lo. Também se procura minimizar a quantidade dos *backlogs* feitos ao longo do tempo b_{it}, já que o custo por *backlog*, p_{it}, é sempre mais alto do que o custo de atendimento (vide definições a seguir), dentro do período, por um servidor mais distante. Finalmente, a função procura minimizar o custo de replicar o conteúdo *k* no servidor *j* a partir do servidor *l* no período *t*, h_{kjlt}.

As restrições (02) associam as variáveis x_{ijt} e b_{it}, indicando que a soma das quantidades entregues num período mais a quantidade em dívida para o período posterior é igual à demanda para o período corrente mais o que ficou em dívida do período anterior. As restrições (03) restringem o fluxo total entregue por um servidor à sua capacidade máxima. As restrições (04) restringem a entrega à banda máxima suportada pelo cliente. As restrições (05) exigem o pleno atendimento a uma requisição. Em (06), condiciona-se que uma requisição só pode ser atendida por servidor que possua réplica do conteúdo demandado. As restrições (07) e (08) controlam a quantidade de réplicas de um conteúdo, estipulando a quantidade mínima de uma réplica durante o tempo de vida do conteúdo e nenhuma fora do tempo de vida. As restrições (09) e (10) fazem com que apenas o servidor origem de um conteúdo o

contenha no período em que ele surge. As restrições (11) garantem que toda replicação crie nova réplica. As restrições (12) exigem que uma replicação só ocorra a partir de um servidor que possua o conteúdo a replicar. As restrições (13) limitam a soma dos tamanhos dos conteúdos em um servidor ao seu espaço em disco. As restrições (14) a (17) são de integralidade e não negatividade. Se a redistribuição de requisições gera uma violação a alguma restrição, a solução é penalizada.

Chamou-se de POP-PPRDR o algoritmo que implementa o *framework* POPMUSIC aplicado ao PPRDR, de acordo com as escolhas livres dos componentes do método POPMUSIC apresentadas nessa seção. No próximo capítulo descrevem-se os experimentos computacionais realizados com o POP-PPRDR.

4. Experimentos Computacionais

Uma heurística baseada no método POPMUSIC para resolver de forma heurística o Problema de Posicionamento de Réplicas e Distribuição de Requisições em RDCs (POP-PPRDR) foi apresentada no capítulo anterior. Neste capítulo são apresentados dois experimentos computacionais envolvendo essa heurística. O primeiro estudo objetiva ajustar o valor do parâmetro que controla o número de iterações da busca local embutida na heurística proposta. O segundo estudo objetiva avaliar a eficiência do algoritmo proposto comparando-o com outro método disponível na literatura. Para a definição do tamanho dos subproblemas avaliados, conduziu-se também experimentos que são brevemente relatados.

4.1 Ambiente computacional e instâncias de teste

Todos os experimentos computacionais relatados foram realizados em um equipamento com processador Intel Core i7-2600, 3,40GHz e 16 GB de memória RAM, rodando Linux 3.2.0-52 ubuntu 12.04 LTS 64-bit. Os algoritmos foram implementados na linguagem C++ e compilados com o compilador g++ v. 4.3.

Com o objetivo de avaliar a qualidade da heurística POP-PPRDR buscou-se considerar problemas testes para os quais houvesse literatura a respeito e que estivessem disponíveis. Desta forma, foram consideradas as instâncias sintetizadas por NEVES (2011). O autor criou quatro classes de instâncias, identificadas por A, B, C e D.

A Classe A contém instâncias em escala reduzida, usadas para testes e com praticamente todos os valores escolhidos de maneira arbitrária. A Classe B contém

instâncias baseadas em valores encontrados na literatura para problemas semelhantes aos abordados na tese de NEVES (2011) e baseadas na realidade de equipamentos e redes disponíveis no mercado ao final do ano de 2008. As instâncias da Classe C são semelhantes às da Classe B, com aumento das restrições quanto à capacidade de armazenamento dos servidores. A Classe D contém instâncias com restrições mais severas na capacidade de armazenamento e na banda dos servidores quando comparada à Classe C. Em testes computacionais, NEVES (2011) constatou que as instâncias das Classes A e B são de fácil resolução. Essas classes não foram consideradas nesta dissertação. NEVES (2011) considerou pouco conclusivos os resultados de seu método aplicado a instâncias com mais de 100 servidores e assim julgou desnecessário criar instâncias da Classe D com tamanho superior a 100. Como o objetivo deste estudo é resolver instâncias de tamanho 100 e maior, comparando os resultados contra os resultados de NEVES (2011), que não existem para a Classe D, essa classe também não foi considerada nesta dissertação, que se limita a instâncias de Classe C. A tabela abaixo relaciona as características das instâncias da Classe C:

Quantidade de Servidores na Instância	Identificação da Instância	Quantidade de Instâncias
10, 20, 30 ou 50	11 até 15	5
100, 200, 300, 400 ou 500	1 até 5	5
600	1	1

Tabela 1. Tamanho e identificação das instâncias da Classe C apresentadas no trabalho de NEVES (2011).

A seguir relata-se o processo de geração de instâncias da Classe C, conforme NEVES (2011). O primeiro passo é a criação da topologia da rede, que foi resolvida com o auxílio da ferramenta BRITE da Boston University. A topologia da rede, que informa quais servidores se comunicam entre si, é considerada um dado para o presente trabalho. No segundo passo, NEVES (2011) determinou a quantidade de períodos de tempo que serão considerados no processo de otimização (35), valor também utilizado neste trabalho. No passo seguinte, foram definidas as distâncias entre os servidores (em termos do tempo que os dados levam para percorrer o caminho de um a outro) no primeiro período de tempo, variando de 60 a 100 milissegundos, dentro do intervalo entre 15 a 300 milissegundos observado nos estudos de GINJUPALLI (2005). Os canais da rede foram definidos como sendo simétricos (o tempo que um pacote de dados leva

para ir de um servidor *j* a um servidor *l* é igual ao tempo gasto para os dados irem do servidor *l* ao *j*).

No quarto passo, foram geradas as informações dos servidores: cada um recebeu um identificador único e, para o espaço em disco de cada servidor, usou-se a distribuição uniforme de probabilidade, com valores determinados experimentalmente, entre 3 e 4 GB; para a banda de cada servidor, também se utilizou a distribuição uniforme de probabilidade, com valores oscilando entre 4.000 e 4.050 MB. Os valores de banda foram escolhidos com base em servidores reais disponíveis no mercado à época da geração das instâncias, conforme NEVES (2011). O próximo passo do processo de geração de instâncias foi a geração das informações dos conteúdos. Os conteúdos foram divididos entre permanentes (permanecem na RDC desde o período inicial até o último período de tempo) e voláteis (surgem fora do primeiro período e podem ser removidos antes do período final). Cada conteúdo recebeu um identificador. Definiu-se a quantidade de conteúdos permanentes a usar como 10. Para a quantidade de conteúdos voláteis, foram escolhidos números aleatórios entre 1 e 5. A quantidade total de conteúdos em cada instância é compatível com o que foi encontrado na literatura (BARTOLINI *et al.*, 2003, BEKTAS *et al.*, 2007). NEVES (2011) não localizou na literatura trabalhos sobre a proporção entre conteúdos permanentes e voláteis e assim a proporção utilizada para as instâncias foi por ele definida. Para o tamanho dos conteúdos, foram utilizados números aleatórios entre 250 e 400 MB. As faixas de tamanhos foram baseadas em vídeos típicos com duração de cerca de 40 minutos disponíveis para *download* na internet. A cada conteúdo foi atribuído, de maneira aleatória, um servidor (tendo todos os servidores a mesma chance de escolha) que passa a ser conhecido como "servidor origem" do conteúdo, ou seja, o servidor da RDC onde o provedor do conteúdo o inseriu.

A sexta etapa foi a geração de requisições. As requisições devem levar em conta informações dos conteúdos, para que não sejam geradas requisições para conteúdos não disponíveis na RDC e para identificar quais os conteúdos mais populares; e dos servidores, pois no momento da geração das requisições cada uma é atrelada a um servidor ao qual o cliente – que gera a requisição – está conectado (o "servidor de origem" da requisição). Foi definido por NEVES (2011) que os conteúdos permanentes eram mais populares do que os conteúdos voláteis e que, dentro de cada grupo de conteúdos, aqueles identificados com os menores números eram os mais populares. O

autor também definiu que a quantidade total de requisições por período de tempo, *numT*, é descrita por *numT* = *rand*(20,25) x |*V*|/5. Decidida a quantidade, deve-se determinar como as requisições são divididas entre os conteúdos. Segundo BRESLAU *et al.* (1999) e WAUTERS *et al.* (2005), as requisições de busca para múltiplas chaves tendem a seguir um padrão semelhante à distribuição Zipf criada por George K. Zipf em seu trabalho "*Relative Frequency as a Determinant of Phonetic Change*" em Harvard Studies in Classical Philology – vol. 40, 1929 (citado pelos autores) que diz que as chaves mais populares tendem a concentrar a imensa maioria da quantidade de requisições. Considerando-se esse princípio e conhecendo quais os conteúdos mais populares, NEVES (2011) decidiu que cada conteúdo recebesse 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 a quantidade total de conteúdos e p_k indica a posição do conteúdo *k* em termos de popularidade.

Sendo *Q* o conjunto de requisições, cada requisição *i* (*i* = 1, 2, ..., |*Q*|) traz associada o atraso máximo permitido TD_i, que varia entre 400 e 800 ms; a banda mínima requerida BR_i, que assume valores entre 40 e 42 MB por período; a banda máxima BX_{it} disponibilizada no período *t*, que varia entre 45 e 50 MB por período; e o atraso local (*ld*(*i*)), que varia entre 50 e 100 ms.

A exemplo do praticado no modelo proposto por NEVES (2011), considerou-se neste trabalho períodos com tamanho específico de 60 segundos, tamanho razoável de acordo com os tempos usados em outros trabalhos (AIOFFI *et al.*, 2005, e WOLFSON *et al.*, 1997). Ainda de acordo com o modelo de NEVES (2011), nos casos de conteúdos grandes, não passíveis de transmissão durante um único período, a demanda por conteúdo foi fracionada e seu atendimento ocorreu ao longo de dois ou mais períodos, por um ou mais servidores.

4.2 Estudo de valores para o parâmetro que controla o critério de parada da busca local

Na implementação de NEVES (2011), o ILSAM (*Iterated Local Search with Adaptive Memory*) foi usado como o principal mecanismo de busca. Neste trabalho, o ILSAM é igualmente usado como mecanismo para tentar melhorar o resultado de subproblemas formados a partir da solução inicial, controlado pelo POPMUSIC. O ILSAM foi

adaptado e calibrado para que a execução de cada rodada não demorasse os 7.200 segundos do trabalho original e sim uma pequena fração de tempo, a fim de que a execução do algoritmo como um todo não se alongasse demasiadamente.

Como o ILSAM alterna entre modos de execução “normal”, “diversificado” e “intensificado” para tentar obter uma melhor solução, escolheu-se limitar o tempo através do controle da quantidade de iterações sem melhora. Conforme descrito na Seção 3.2, NEVES (2011) estabeleceu um limite de iterações sem melhora para alternar entre os diferentes estados, e determinou experimentalmente o limite como sendo 10. Isso significa que, estando em qualquer estado, após 10 iterações sem melhora, o algoritmo muda de estado. O comportamento é ilustrado na Figura 8. Supondo que o algoritmo esteja no estado de intensificação I; ao iterar 10 vezes sem melhora, ele vai para o estado normal N. De N, 10 rodadas sem melhora levam ao estado de diversificação D1. De D1, 10 iterações sem melhora conduzem ao estado de diversificação D2, mais ampla do que D1. De D2, 10 rodadas sem melhora levam a N.

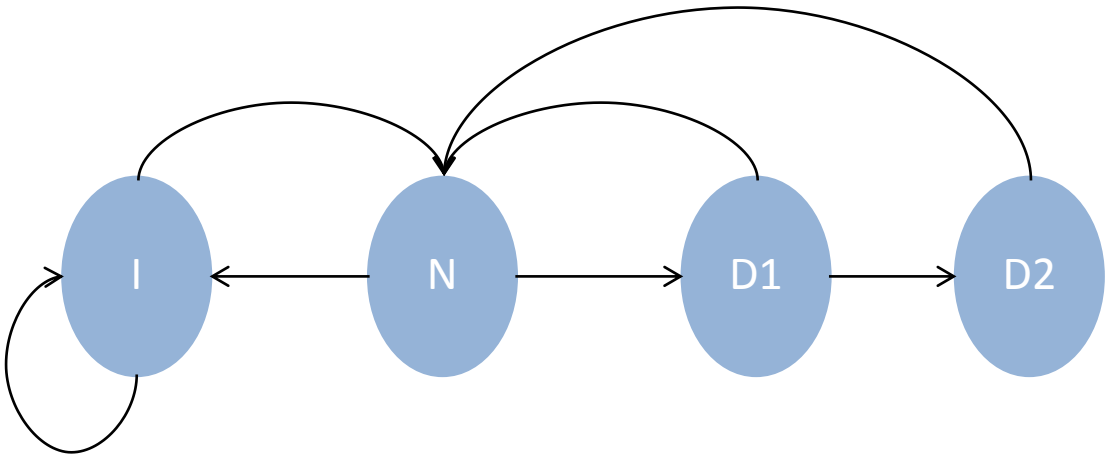

Figura 8. Diagrama de Estados do ILSAM segundo NEVES (2011).

Foram testadas as quantidades de 30, 40 e 70 iterações no ILSAM sem melhora como critério de parada, correspondentes respectivamente aos seguintes esquemas:

Quantidade de iterações sem melhora = 30. Ciclo simples sem melhora, não considerando o estado de intensificação, definido da seguinte forma:

- início no estado N, em seguida 10 rodadas sem melhora conduzem a D1; em D1, 10 rodadas sem melhora conduzem a D2; em D2, 10 rodadas sem melhora retornam a N.

Quantidade de iterações sem melhora = 40. Ciclo completo sem melhora, considerando todos os quatro estados, definido da seguinte forma:

- início no estado I, em seguida 10 rodadas sem melhora conduzem a N; em N, 10 rodadas sem melhora conduzem a D1; em D1, 10 rodadas sem melhora conduzem a D2; por fim, em D2, 10 rodadas sem melhora retornam a N.

Quantidade de iterações sem melhora = 70. Ciclo completo sem melhora, considerando os quatro estados e duas voltas pelos estados Normal–Diversificação, definido da seguinte forma:

- início no estado I, em seguida 10 rodadas sem melhora conduzem a N; em N, 10 rodadas sem melhora conduzem a D1; em D1, 10 rodadas sem melhora conduzem a D2; em D2, 10 rodadas sem melhora retornam a N; em N, 10 rodadas sem melhora conduzem a D1; em D1, 10 rodadas sem melhora conduzem a D2; por fim, em D2, 10 rodadas sem melhora retornam a N.

O estudo de valores para o parâmetro que controla o critério de parada da busca local foi realizado para duas instâncias, identificadas por: Instância 13 com 30 servidores e Instância 1 com 100 servidores. Neste estudo o valor do parâmetro *r* é igual a 6. A Tabela 2 apresenta os resultados obtidos pelo algoritmo POP-PPRDR para a Instância 13 com 30 servidores considerando-se três valores para o limite de iterações sem melhora do ILSAM: 30, 40 e 70. Para cada instância e para cada valor de limite foram realizadas cinco execuções do algoritmo com diferentes sementes para o gerador de números aleatórios usado no ILSAM (os números primos 3, 5, 7, 11 e 13). Na Tabela 2, a linha "Média F.O." apresenta os valores médios da função objetivo; a linha "Tempo Médio" apresenta o tempo médio em segundos necessário para obter os valores da linha anterior; a linha "Melhor F.O." apresenta o melhor valor da função objetivo encontrado entre as cinco execuções e, por fim, a linha "Tempo" apresenta o tempo em segundos necessário para obter o melhor valor da função objetivo, mostrado na linha acima.

	Iterações sem melhora no ILSAM		
	Limite = 30	Limite = 40	Limite = 70
Média F.O.	6.618.197	6.617.097	6.616.961
Tempo médio	2.137	2.934	5.580
Melhor F.O.	6.618.165	6.615.889	6.616.087
Tempo	1.788	3.025	5.644

Tabela 2. Resultados POP-PPRDR limitando a quantidade de iterações sem melhora no ILSAM a 30, 40 e 70, para a Instância 13 com 30 servidores.

A variação entre os resultados apresentados na Tabela 2 está resumida na Tabela 3. Diferenças absolutas e relativas positivas, de um valor de limite para o outro, indicam que o segundo valor de limite é melhor.

	Diferenças entre os valores obtidos por POP-PPRDR			
	De Limite = 30 para 40		De Limite = 40 para 70	
	Diferença absoluta	Diferença relativa (%)	Diferença absoluta	Diferença relativa (%)
Média F.O.	1.130	0,02	106	0,00
Tempo médio	-797	-37,28	-2.646	-90,17
Melhor F.O.	2.276	0,03	-198	0,00
Tempo	-1.237	-69,17	-2.619	-86,58

Tabela 3. Resumo da variação dos resultados apresentados na Tabela 2, para a Instância 13 com 30 servidores.

Quando o limite de iterações muda de 30 para 40, observa-se uma melhora irrisória no resultado médio (0,02%). No entanto, o aumento da quantidade de iterações provoca também aumento no tempo dos laços de busca local, levando a um aumento no tempo de execução do POP-PPRDR da ordem de 37%. O tempo para atingir o melhor resultado com Limite=40, que foi apenas 0,03% melhor do que com Limite=30, piorou 69%.

Com a mudança no limite de iterações de 40 para 70, não se observa ganho até a segunda casa decimal no resultado médio, embora o aumento no tempo de execução do algoritmo POP-PPRDR tenha sido significativo (90%). Avaliação similar ocorre para o melhor resultado.

A Tabela 4, de formato similar à Tabela 2, mostra os resultados do algoritmo POP-PPRDR para a Instância 1 com 100 servidores, em execuções com limite de iterações sem melhora definidos como 40 e 70 no ILSAM.

	Iterações sem melhora no ILSAM	
	Limite = 40	Limite = 70
Média F.O.	23.703.208	23.690.446
Tempo médio	17.928	35.195
Melhor F.O.	23.693.279	23.688.115
Tempo	19.134	34.117

Tabela 4. Resultados POP-PPRDR limitando a quantidade de iterações sem melhora no ILSAM a 40 e 70, para a Instância 1 com 100 servidores.

Para instância de tamanho maior, o aumento do limite de iterações traz um ganho maior obtido no resultado médio, em termos absolutos (12.761); percentualmente, no entanto, o ganho continua pequeno (0,05%), para um aumento no tempo de execução do algoritmo POP-PPRDR de 96%. Avaliação similar ocorre para o melhor resultado. A variação entre os resultados está resumida na Tabela 5.

	Diferenças entre os valores obtidos por POP-PPRDR	
	De Limite = 40 para 70	
	Diferença absoluta	Diferença relativa (%)
Média F.O.	12.761	0,05
Tempo médio	-17.266	96,31
Melhor F.O.	5.164	0,02
Tempo	-14.983	78,31

Tabela 5. Resumo da variação dos resultados apresentados na Tabela 4, para a Instância 1 com 100 servidores.

Observando-se os resultados do estudo apresentado nesta seção nota-se que o limite com valor igual a 30 gera resultados discretamente piores, e embora poupando tempo razoável, não contempla todos os estados possíveis do ILSAM; e que o valor de 70 foi capaz de gerar resultados discretamente melhores, quase dobrando, no entanto, os tempos de execução. Desta forma, decidiu-se usar neste trabalho o valor de 40 como limite de execuções sem melhora no ILSAM, por permitir um ciclo completo ao longo de todos os estados.

4.3 Estudo de valores para o parâmetro *r*

O único parâmetro do método POPMUSIC é *r*, a quantidade de partes usadas para formar um subproblema. De acordo com ALVIM e TAILLARD (2013), se os subproblemas são pequenos demais, pode ocorrer de não se alcançar melhorias; se grandes demais, as melhorias podem demandar um esforço computacional proibitivo. Entretanto, a escolha do valor de *r* deve ser feita de forma experimental. Nesta seção investiga-se o comportamento do Algoritmo POP-PPRDR utilizando diferentes valores de *r*. Neste estudo o valor do parâmetro que controla o critério de parada da busca local ILSAM é igual a 40. Igualmente ao estudo anterior foram realizadas cinco execuções do Algoritmo POP-PPRDR com diferentes valores para a semente (valores iguais a 3, 5, 7, 11 e 13) do gerador de números aleatórios utilizado na busca local ILSAM.

As Tabelas 6, 7 e 8 apresentam um resumo dos principais resultados. Cada linha da tabela corresponde aos valores médios obtidos com um determinado tamanho de subproblema, apresentado na primeira coluna (*r*). A linha correspondente a *r* = 6 está marcada em cinza como referência. A segunda coluna apresenta o valor médio da função objetivo alcançado ao longo das execuções; a terceira coluna, "*Gap* relativo a *r*=6", apresenta a variação (*gap*) percentual entre o valor médio obtido para *r* = 6 e o valor médio obtido para o *r* em questão. A quarta coluna apresenta o tempo médio (em segundos) decorrido para alcançar o final da execução com o *r* indicado, e a coluna "% relativo a *r*=6" indica o percentual que esse tempo representa com relação ao tempo médio para alcançar o final da execução para *r* = 6. A média do melhor valor obtido para a função objetivo, e o respectivo tempo médio em que o valor foi alcançado, estão marcados em azul.

Para a Instância 13 de tamanho 30, o melhor valor da função objetivo foi obtido para *r* = 5. O segundo melhor valor foi obtido para *r* = 7 e o terceiro para *r* = 6. O valor da função objetivo piorou tanto para valores de *r* menores do que 6 quanto para valores maiores do que 6, afastando-se de *r* = 6. Os resultados estão registrados na Tabela 6.

r	Média F.O.	*Gap* relativo a *r*=6	Tempo (seg.)	% relativo a *r*=6
3	6.618.899	+ 0,027677 %	1.027	35,01
4	6.617.237	+ 0,002570 %	1.612	54,92
5	6.616.689	- 0,005707 %	2.049	69,81
6	6.617.067		2.934	
7	6.616.903	- 0,002481 %	3.248	110,69
8	6.617.714	+ 0,009782 %	3.919	133,54
9	6.617.837	+ 0,011628 %	4.649	158,45

Tabela 6. Variação nos resultados do Algoritmo POP-PPRDR para diferentes tamanhos de subproblemas. Instância 13 com 30 servidores.

Para a Instância 1 de tamanho 100, o melhor valor da função objetivo foi obtido para *r* = 9. O segundo melhor foi obtido para *r* = 8 e o terceiro para *r* = 6. O valor da função objetivo piorou progressivamente com a diminuição do valor de *r* a partir do valor 6 e também para *r* = 7, maior do que 6. Considerando a melhora obtida no valor da função objetivo com *r* = 8 e, ainda mais, com *r* = 9, decidiu-se testar *r* = 10, verificando-se que o valor da função objetivo voltou a subir, e o tempo de execução foi bastante maior. Os resultados estão registrados na Tabela 7.

r	Média F.O.	*Gap* relativo a *r*=6	Tempo (seg.)	% relativo a *r*=6
3	23.730.218	+ 0,113954 %	4.230	23,59
4	23.722.853	+ 0,082879 %	7.497	41,82
5	23.707.396	+ 0,017669 %	12.727	70,99
6	23.703.208		17.928	
7	23.704.083	+ 0,003693 %	22.856	127,49
8	23.696.293	- 0,029172 %	30.708	171,28
9	23.694.233	- 0,037863 %	36.482	203,49
10	23.695.773	- 0,031367 %	47.357	264,15

Tabela 7. Variação nos resultados do Algoritmo POP-PPRDR para diferentes tamanhos de subproblemas. Instância 1 com 100 servidores.

Para a instância de tamanho 400, foram executados testes com as cinco sementes para apenas três valores de *r*: 3, 6 e 9. A decisão foi devida a dois fatores: primeiro, a intenção dos testes era identificar se havia um determinado valor de *r* que gerasse o melhor resultado para todos os tamanhos de instância, o que se verificou não ser verdade durante os testes com as instâncias de tamanho 30 e 100; e segundo, os testes com a instância 400 visaram principalmente observar se havia uma tendência de melhoria de resultados conforme r variasse para mais ou para menos. O que se verificou ao final dos testes com os três valores de *r* foi que não houve tal tendência; o melhor valor da função objetivo foi obtido para *r* = 3, o valor da função objetivo piorou conforme *r* cresceu de 3 para 6, e melhorou com novo aumento de *r* (*r* = 9). Os resultados estão registrados na Tabela 8.

r	Média F.O.	*Gap* relativo a *r*=6	Tempo (seg.)	% relativo a *r*=6
3	19.717.328.483	- 0,000153 %	18.447	37,94
6	19.717.358.709		48.626	
9	19.717.332.972	- 0,000131 %	109.514	225,22

Tabela 8. Variação nos resultados do Algoritmo POP-PPRDR para diferentes tamanhos de subproblemas. Instância 1 com 400 servidores.

Em todos os testes, verificou-se que o tempo computacional total do algoritmo cresce com o tamanho dos subproblemas tratados. No entanto, não foi possível observar uma tendência clara para a relação entre valor da função objetivo e *r*. Também não foi encontrado um mesmo valor de *r* que gerasse o melhor valor da função objetivo em todos os tamanhos de instância. Por isso, decidiu-se usar neste trabalho o valor intermediário de *r* = 6 para o tamanho dos subproblemas.

4.4 Comparação de POP-PPRDR contra o método ORIGINAL

Nesta seção comparam-se os resultados da aplicação de dois métodos para resolver o PPRDR: o método proposto por NEVES (2011) e identificado por ORIGINAL e o algoritmo proposto POP-PPRDR, baseado na metaheurística POPMUSIC e que usa o procedimento ILSAM como método de busca embutido. Todos os testes foram executados no mesmo ambiente computacional, descrito na Seção 4.1. Para cada instância examinada, o algoritmo escolhido foi executado cinco vezes, uma para cada diferente semente do gerador de números aleatórios usado na busca local ILSAM (números 3, 5, 7, 11 e 13). Os tempos de execução são expressos em segundos.

O Algoritmo ORIGINAL foi executado para as instâncias da classe C, conforme definidas na Seção 4.1 (ao todo, 180 execuções). Para cada execução foi estabelecido um limite de tempo, conforme NEVES (2011): uma iteração teria que começar em até 7.199 segundos contados do início daquela execução.

As execuções do POP-PPRDR obedeceram ao limite máximo de 40 iterações sem melhora no laço de busca local, com o valor de *r* definido como 6. O critério de parada foi a inexistência de subproblemas a otimizar.

As Tabelas 9 e 10 apresentam os resultados médios de cinco execuções obtidos, respectivamente, para as instâncias 13, com 30 servidores, e 15, com 30 servidores. Nas duas tabelas, a coluna ORIGINAL mostra os resultados obtidos executando a versão original do algoritmo proposto por NEVES (2011). A coluna POP-PPRDR apresenta os valores obtidos pelo algoritmo proposto usando POPMUSIC, e as duas últimas colunas exibem as diferenças absoluta e relativa entre os dois métodos, ORIGINAL e POP-PPRDR. Diferenças positivas indicam melhorias. Na primeira linha são mostradas as médias dos valores da função objetivo, e na segunda linha são apresentadas as médias dos tempos de execução dos algoritmos, em segundos. Na terceira linha apresenta-se o melhor valor alcançado para a função objetivo durante as execuções e na linha seguinte o tempo em segundos para alcançá-lo.

	ORIGINAL (a)	POP-PPRDR (b)	Diferença (a-b)	Diferença %
Média F.O.	6.630.096	6.617.067	13.029	0,20
Tempo médio	7.212	2.934	3.420	53,82
Melhor F.O.	6.628.942	6.615.889	13.053	0,20
Tempo	7.210	3.025	4.185	58,05

Tabela 9. Resultados dos algoritmos ORIGINAL e POP-PPRDR. Instância 13 com 30 servidores.

	ORIGINAL (a)	POP-PPRDR (b)	Diferença (a-b)	Diferença %
Média F.O.	6.527.669	6.520.131	7.538	0,12
Tempo médio	7.214	2.237	2.351	51,24
Melhor F.O.	6.527.017	6.518.264	8.753	0,13
Tempo	4.843	2.092	2.750	56,80

Tabela 10. Resultados dos algoritmos ORIGINAL e POP-PPRDR. Instância 15 com 30 servidores.

Observa-se nas Tabelas 10 e 11 que o POP-PPRDR atingiu um valor médio melhor para a função objetivo, quando comparado aos resultados de ORIGINAL, e esse valor médio foi melhor do que o melhor resultado de ORIGINAL. Os tempos para alcançar as soluções e para alcançar a melhor solução foram menores em POP-PPRDR do que em ORIGINAL. A melhora média atingida pelo POP-PPRDR foi 2,98 vezes melhor do que a obtida pelo ORIGINAL. O consumo de tempo de POP-PPRDR foi de apenas 47,26% do tempo de ORIGINAL. Em termos do melhor valor alcançado, POP-PPRDR consumiu apenas 38,33% do tempo utilizado por ORIGINAL e foi 2,79 vezes melhor do que ORIGINAL.

A Figura 9 ilustra o desempenho dos dois algoritmos processando a Instância 13 com 30 servidores.

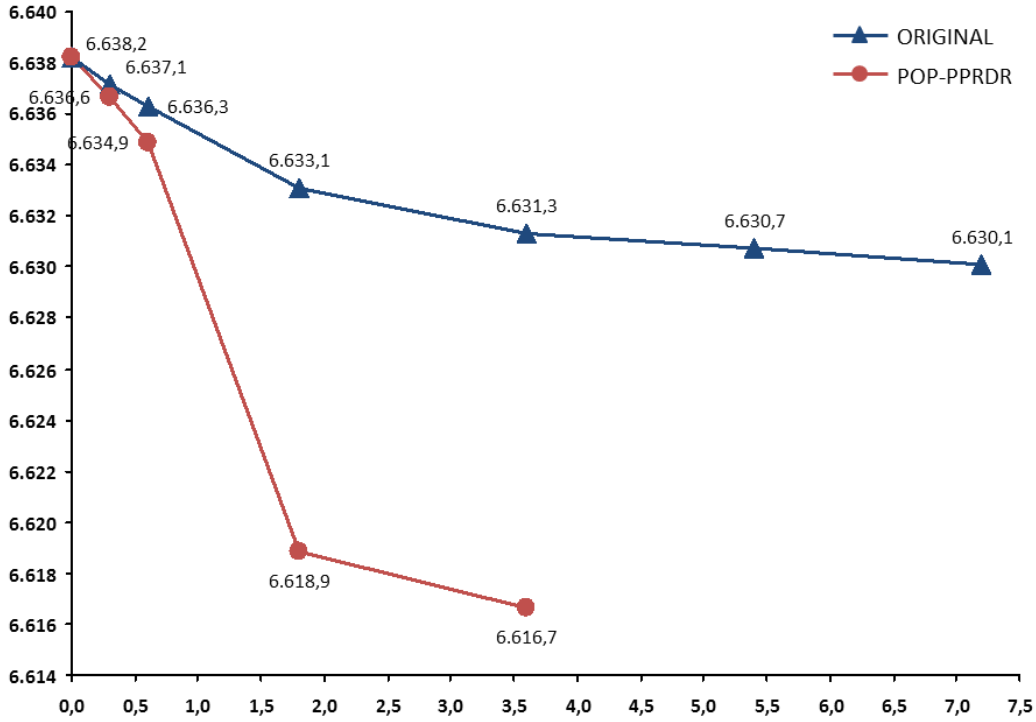

Figura 9. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR. Instância 13 com 30 servidores. Curvas e eixo Y: valores expressos em milhares; eixo X: tempo expresso em mil segundos.

Todas as instâncias da Classe C com 100 servidores foram testadas. Considerando os diferentes critérios de parada dos algoritmos, decidiu-se comparar os resultados de POP-PPRDR nas proximidades de marcos de tempo fixos, estabelecidos em ORIGINAL: 300, 600, 1.800, 3.600, 5.400 e 7.200 segundos, registrando a média das execuções com cada uma das cinco sementes. Observando os resultados nas proximidades do marco de 7.200 segundos (7.194 segundos na média das execuções), por exemplo, POP-PPRDR obteve uma melhora média 151 vezes melhor do que a média obtida por ORIGINAL, considerando o desempenho consolidado de cada um dos dois algoritmos para as cinco instâncias da Classe C (de 1 a 5) com 100 servidores e para as cinco sementes do gerador de números aleatórios usado na busca local ILSAM. O melhor valor alcançado pelo algoritmo POP-PPRDR foi 56 vezes melhor do que o atingido pelo ORIGINAL. Para isso, consumiu 4,65 vezes mais tempo. O tempo médio total de execução do POP-PPRDR, até não haver mais subproblemas a otimizar, foi 2,17 vezes maior do que ORIGINAL, em torno dos 7.200 segundos. Na média, a melhora obtida pelo POP-PPRDR para as instâncias da Classe C com 100 servidores foi 162 vezes melhor do que a obtida pelo algoritmo ORIGINAL, consumindo para isso 2,64 vezes mais tempo do que ORIGINAL.

A Tabela 11 e as Figuras 10 e 11 expõem os resultados obtidos pelos dois algoritmos para a Instância 3, com 100 servidores. Na tabela, a primeira linha explicita os marcos de tempo definidos em ORIGINAL (colunas de 300 a 7.200 segundos), e a segunda linha expõe a média dos tempos próximos a tais marcos alcançados por POP-PPRDR. A terceira linha mostra os valores obtidos por ORIGINAL, registrados em cada marco de tempo, e a quarta linha exibe os valores obtidos por POP-PPRDR nos tempos indicados. A quinta linha apresenta a melhora obtida pelo Algoritmo POP-PPRDR em termos percentuais.

ORIGINAL	300 seg.	600 seg.	1.800 seg.	3.600 seg.	5.400 seg.	7.200 seg.
POP-PPRDR	293 seg.	596 seg.	1.788 seg.	3.605 seg.	5.399 seg.	7.174 seg.
F.O. ORIGINAL	237.015.244	237.015.244	237.015.244	237.015.244	237.015.010	237.015.010
F.O. POP-PPRDR	237.011.698	237.009.996	236.995.850	236.975.056	236.951.240	236.939.368
Melhora %	0,001496	0,002214	0,008183	0,016956	0,026905	0,031914

Tabela 11. Resultados dos algoritmos ORIGINAL e POP-PPRDR. Instância 3 com 100 servidores.

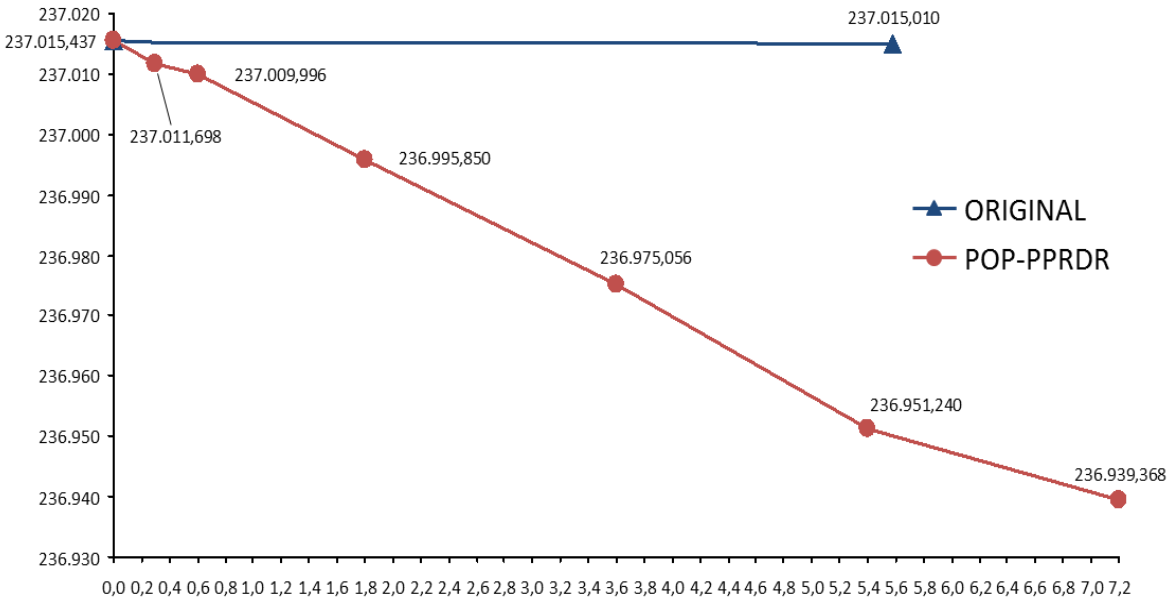

Figura 10. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até 7.200 segundos. Instância 3 com 100 servidores. Curvas e eixo Y: valores expressos em milhares; eixo X: tempo expresso em mil segundos.

A Figura 11 apresenta, além de ORIGINAL, a execução de POP-PPRDR até sua finalização. A melhora obtida com relação ao valor inicial da função objetivo foi de 0,062600 %.

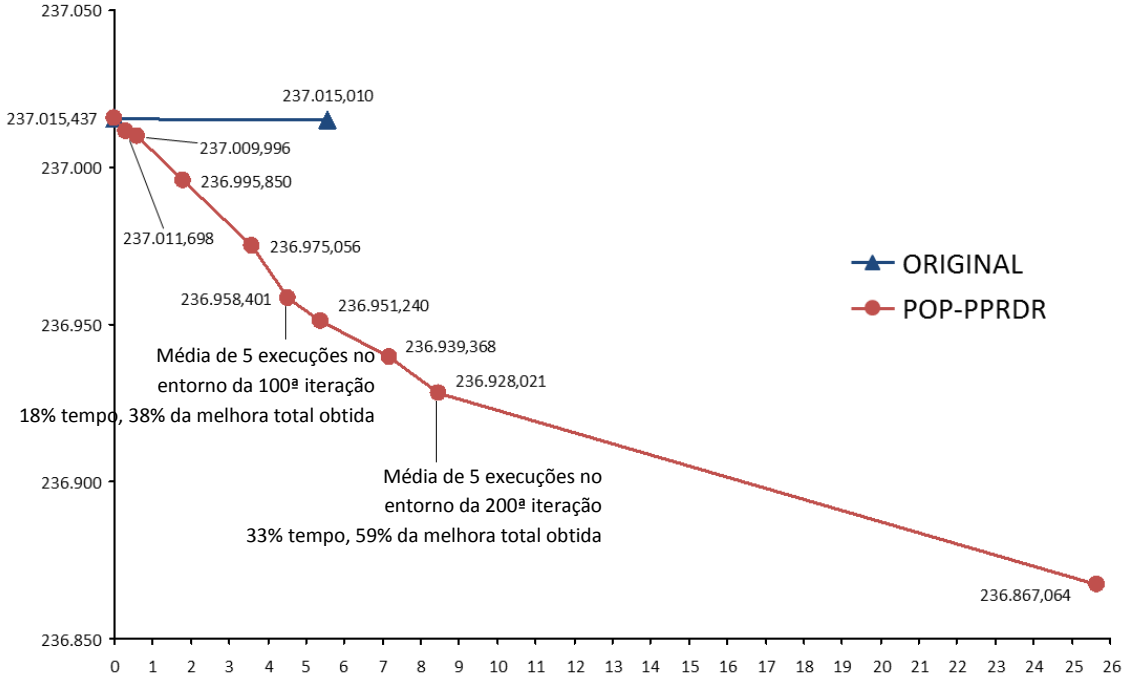

Figura 11. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até parar. Instância 3 com 100 servidores. Curvas e eixo Y: valores expressos em milhares; eixo X: tempo expresso em mil segundos.

Para instâncias com mais de 100 servidores, o trabalho de NEVES (2011) não obtém resultados conclusivos. Em parte das execuções realizadas no presente trabalho, o algoritmo ORIGINAL também não obteve resultados conclusivos; no entanto, em outros casos, foram obtidos resultados que foram então registrados. Nas tabelas numeradas de 12 a 16 e figuras numeradas de 12 a 21, são mostrados dados do desempenho do(s) algoritmo(s) para instâncias com 200, 300, 400, 500 e 600 servidores.

Para a Instância 1 com 200 servidores, ORIGINAL não foi capaz de melhorar a solução inicial no prazo de 7.200 segundos (Figura 12); gerou uma melhora discreta no entorno de 9.000 segundos, e apenas com uma das cinco sementes (Figura 13).

ORIGINAL	300 seg.	600 seg.	1.800 seg.	3.600 seg.	5.400 seg.	7.200 seg.
POP-PPRDR	270 seg.	624 seg.	1.766 seg.	3.668 seg.	5.388 seg.	7.287 seg.
F.O. POP-PPRDR	2.001.956.208	2.001.955.454	2.001.953.544	2.001.949.903	2.001.940.260	2.001.927.824

Tabela 12. Resultados do algoritmo POP-PPRDR. Instância 1 com 200 servidores.

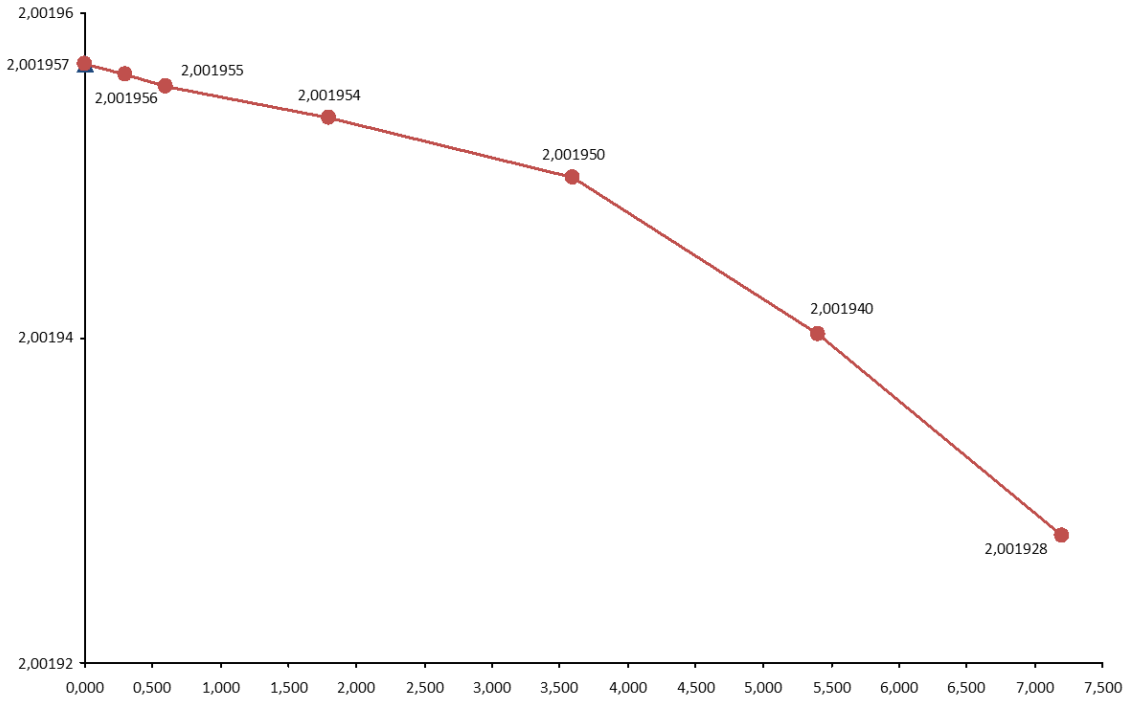

Figura 12. Resultados de execução do Algoritmo POP-PPRDR até 7.200 segundos. Instância 1 com 200 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

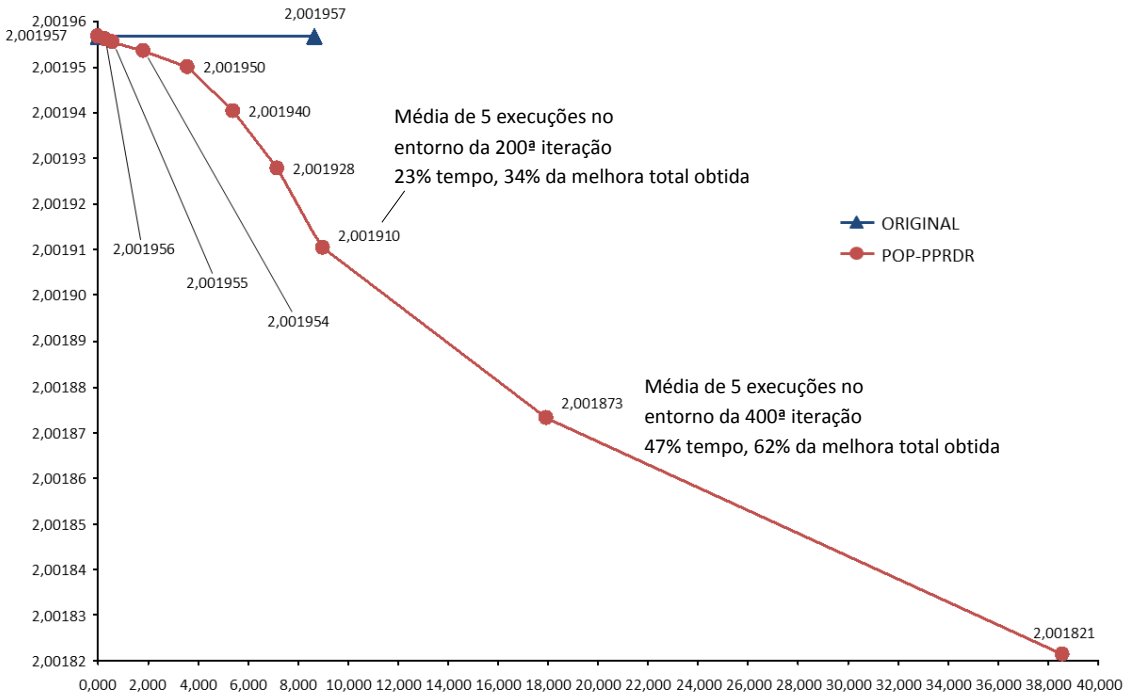

Figura 13. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até parar. Instância 1 com 200 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

Para a Instância 1 com 300 servidores, ORIGINAL não foi capaz de melhorar a solução inicial (Figuras 14 e 15).

ORIGINAL	300 seg.	600 seg.	1.800 seg.	3.600 seg.	5.400 seg.	7.200 seg.
POP-PPRDR	336 seg.	641 seg.	1.743 seg.	3.575 seg.	5.353 seg.	7.214 seg.
F.O. POP-PPRDR	6.091.047.461	6.091.045.085	6.091.043.682	6.091.041.864	6.091.040.120	6.091.036.715

Tabela 13. Resultados do Algoritmo POP-PPRDR. Instância 1 com 300 servidores.

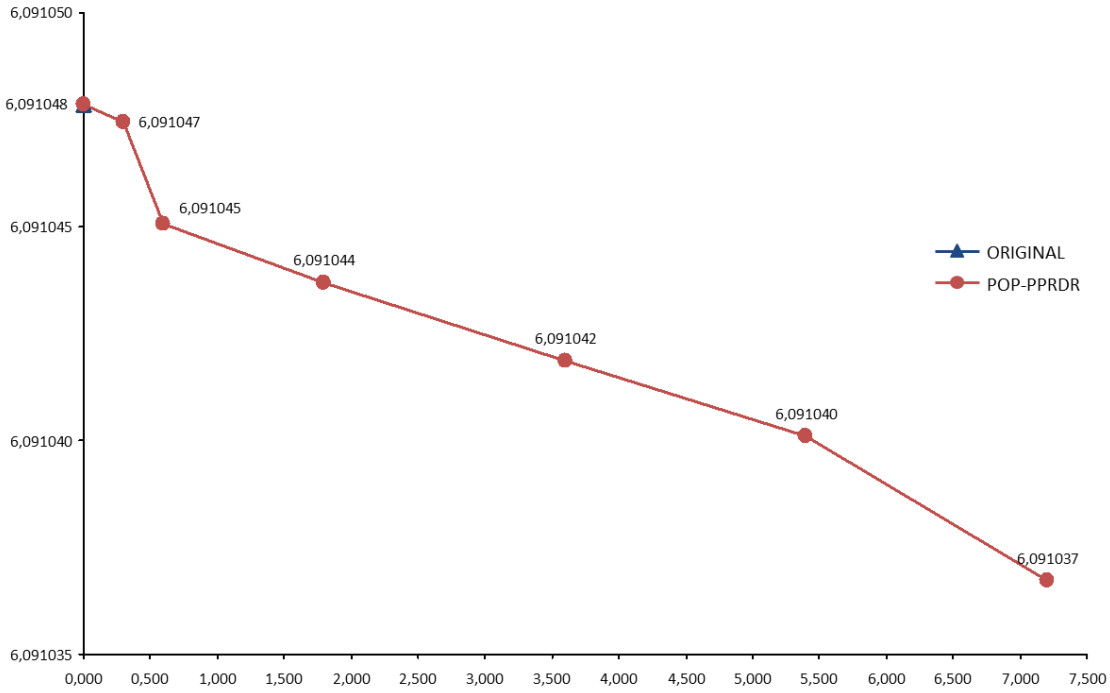

Figura 14. Resultados de execução do Algoritmo POP-PPRDR até 7.200 segundos. Instância 1 com 300 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

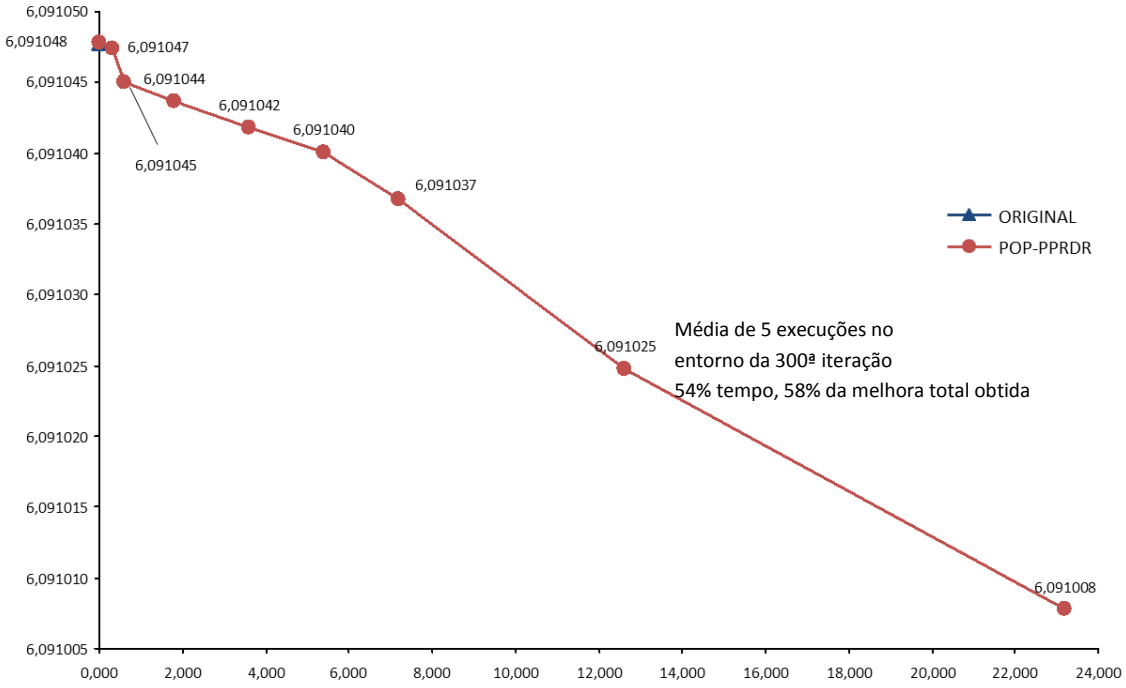

Figura 15. Resultados de execução do Algoritmo POP-PPRDR até parar. Instância 1 com 300 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

Para a Instância 1 com 400 servidores, ORIGINAL não foi capaz de melhorar a solução inicial no prazo de 7.200 segundos (Figura 16); no entanto, gerou resultado cerca de 2,5% melhor do que POP-PPRDR próximo aos 12.000 segundos (Figura 17).

ORIGINAL	300 seg.	600 seg.	1.800 seg.	3.600 seg.	5.400 seg.	7.200 seg.
POP-PPRDR	n/a	1.100 seg.	1.836 seg.	3.722 seg.	5.198 seg.	7.207 seg.
F. O. POP-PPRDR	n/a	19.717.437.903	19.717.437.565	19.717.435.889	19.717.434.538	19.717.430.726

Tabela 14. Resultados do algoritmo POP-PPRDR. Instância 1 com 400 servidores.

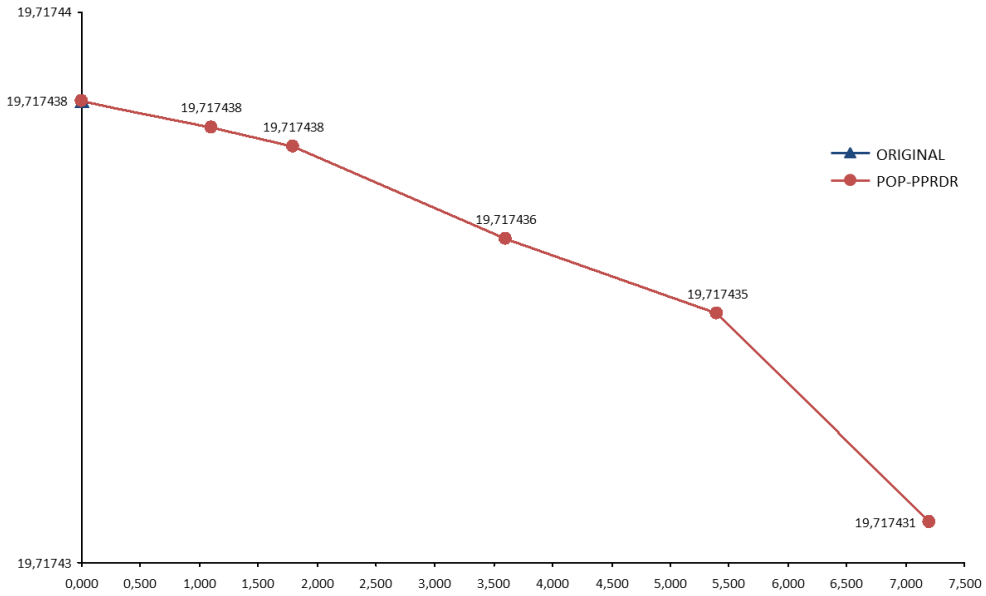

Figura 16. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até 7.200 segundos. Instância 1 com 400 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

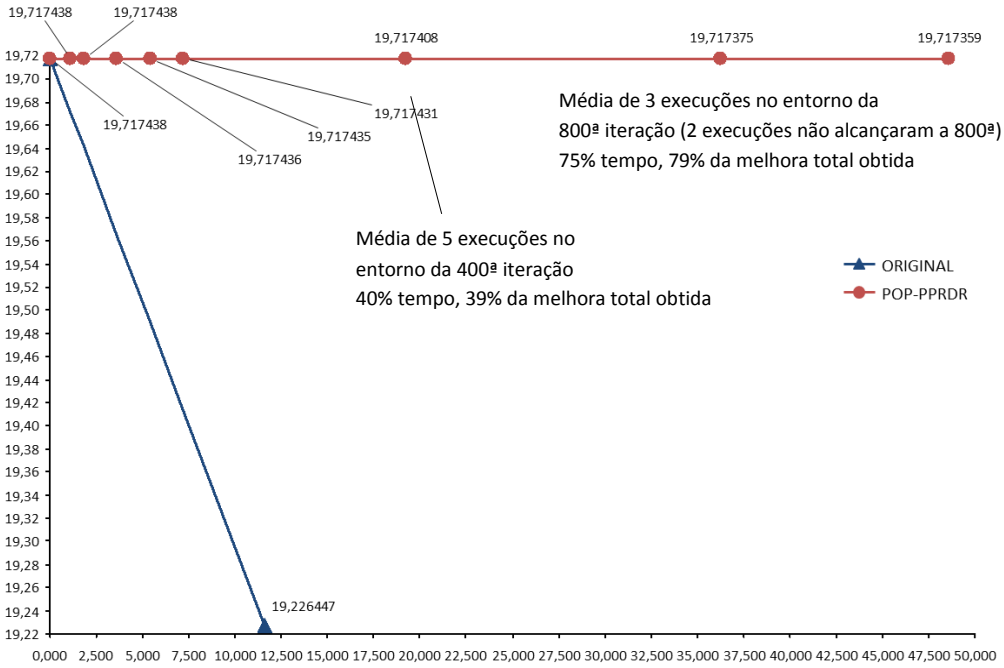

Figura 17. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até parar. Instância 1 com 400 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

Para a Instância 1 com 500 servidores, ORIGINAL não foi capaz de melhorar a solução inicial (Figuras 18 e 19).

ORIGINAL	300 seg.	600 seg.	1.800 seg.	3.600 seg.	5.400 seg.	7.200 seg.
POP-PPRDR	336 seg.	641 seg.	1.743 seg.	3.575 seg.	5.353 seg.	7.214 seg.
F.O. POP-PPRDR	6.091.047.461	6.091.045.085	6.091.043.682	6.091.041.864	6.091.040.120	6.091.036.715

Tabela 15. Resultados do Algoritmo POP-PPRDR. Instância 1 com 500 servidores.

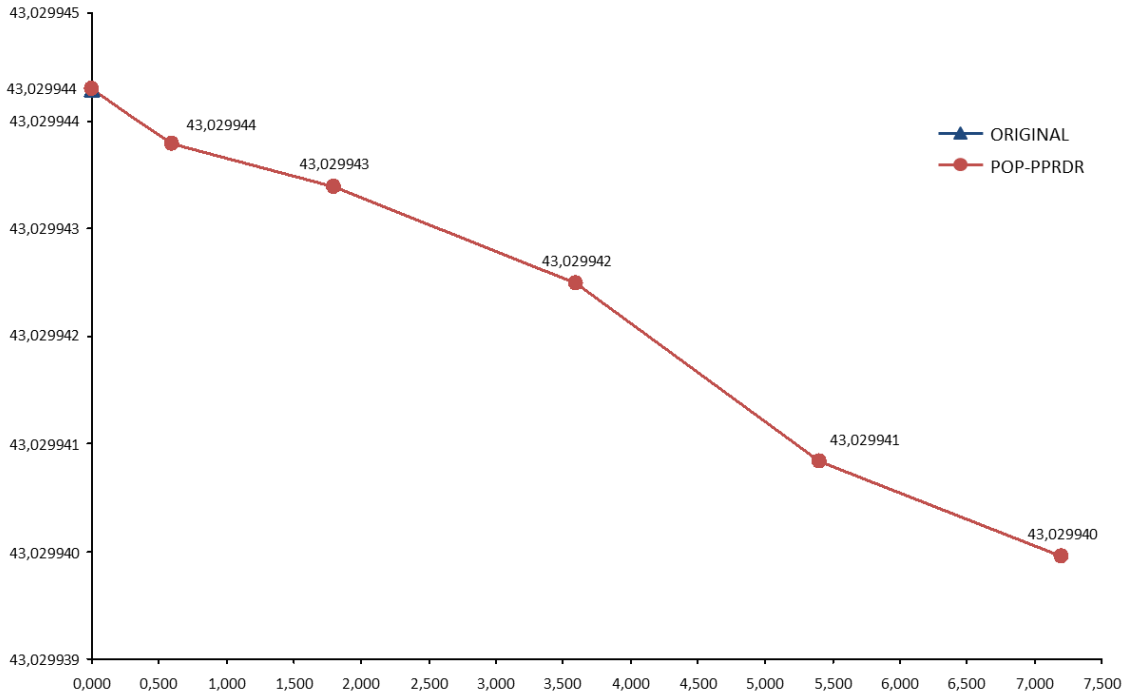

Figura 18. Resultados de execução do Algoritmo POP-PPRDR até 7.200 segundos. Instância 1 com 500 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

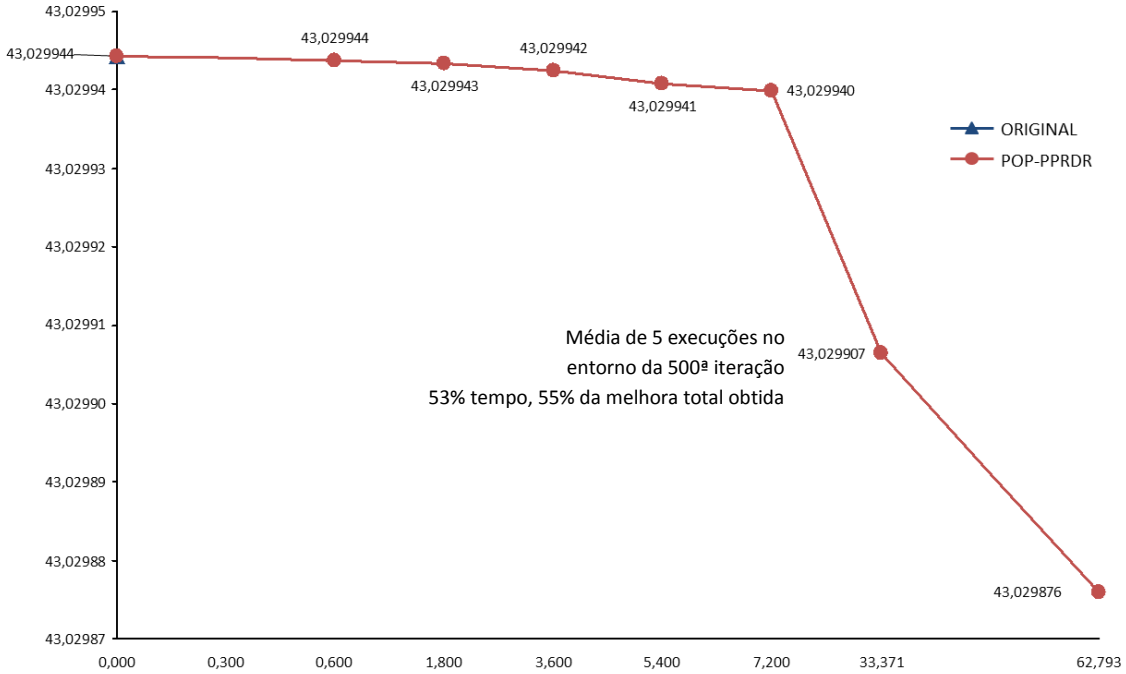

Figura 19. Resultados de execução do Algoritmo POP-PPRDR até parar. Instância 1 com 500 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

Para a Instância 1 com 600 servidores, ORIGINAL não foi capaz de melhorar a solução inicial no prazo de 7.200 segundos (Figura 20); no entanto, gerou resultado cerca de 4% melhor do que POP-PPRDR próximo aos 29.000 segundos (Figura 21).

ORIGINAL	300	600	1.800	3.600	5.400	7.200
POP-PPRDR	n/a	1.081	1.720	4.086	5.647	7.318
F.O. POP-PPRDR	n/a	36.113.406.610	36.113.406.405	36.113.405.436	36.113.404.899	36.113.402.840

Tabela 16. Resultados do Algoritmo POP-PPRDR. Instância 1 com 600 servidores.

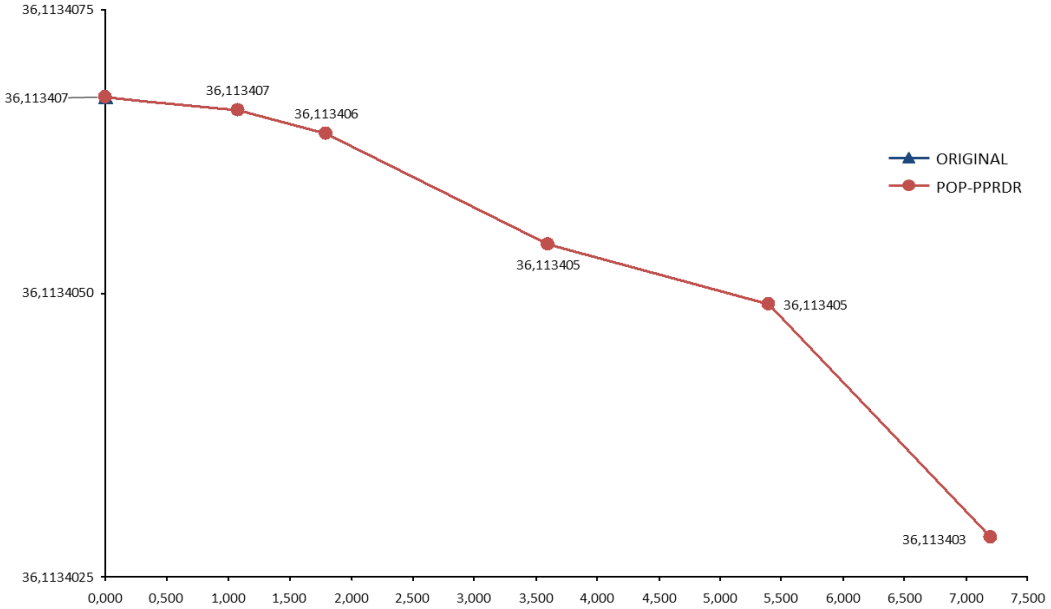

Figura 20. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até 7.200 segundos. Instância 1 com 600 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

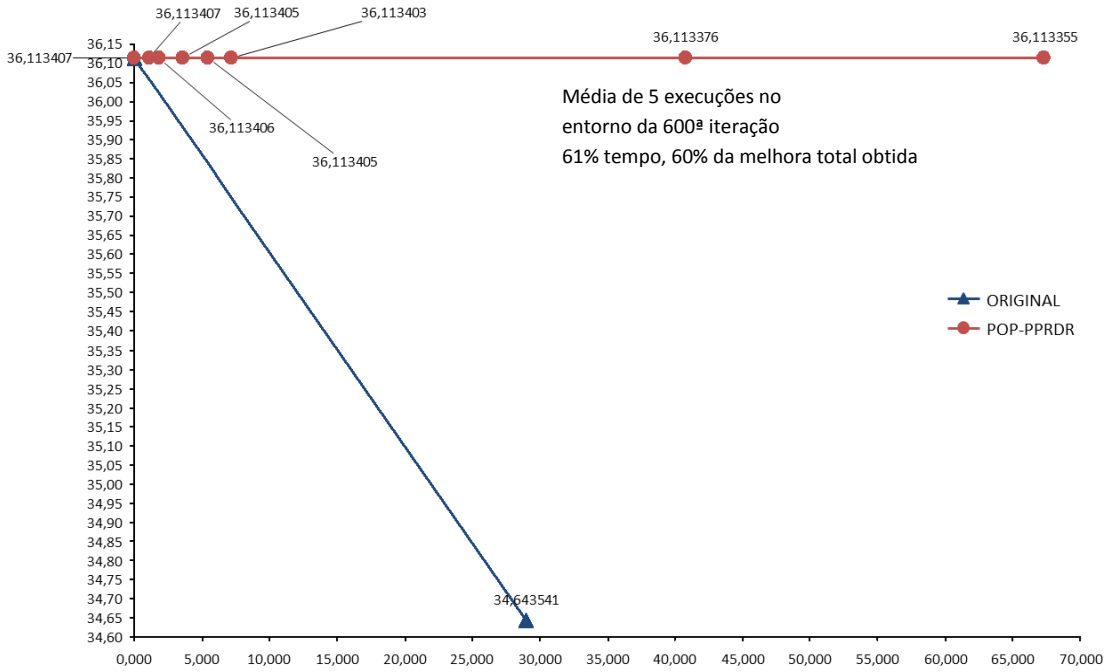

Figura 21. Resultados de execução dos algoritmos ORIGINAL e POP-PPRDR até parar. Instância 1 com 600 servidores. Curvas e eixo Y: valores expressos em bilhões; eixo X: tempo expresso em mil segundos.

Verifica-se no conjunto de gráficos acima que para as instâncias a partir de 300 servidores, o percentual de melhoria obtida passa a ser aproximadamente igual ao percentual de tempo de processamento decorrido em relação ao total: na Figura 15, 300 servidores: em 54% do tempo total, 58% da melhora obtida; na Figura 17, 400 servidores: em 40% do tempo total, 39% da melhora obtida, e em 75% do tempo total, 79% da melhora obtida; na Figura 19, 500 servidores: em 53% do tempo total, 55% da melhora obtida; e na Figura 21, 600 servidores: em 61% do tempo total, 60% da melhora obtida. Esse comportamento permite supor que a melhora obtida será proporcional ao tempo decorrido.

Na Figura 22 são apresentados os tempos utilizados pelo POP-PPRDR para o tratamento de instâncias de tamanho 30, 100, 200, 300, 400, 500 e 600.

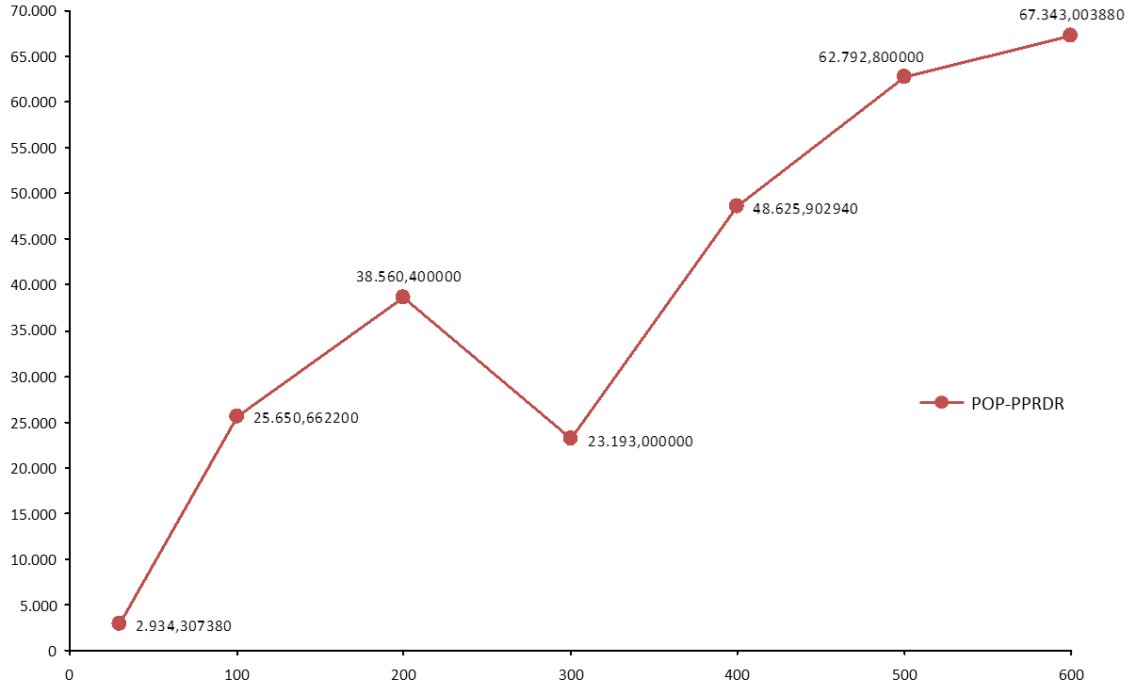

Figura 22. Resultados de execução do Algoritmo POP-PPRDR até parar em função do tamanho da instância. Curvas e eixo Y: tempo expresso em segundos; eixo X: quantidade de servidores na instância.

Os resultados apresentados demonstraram que a aplicação de POPMUSIC ao PPRDR com a metaheurística ILSAM utilizada por NEVES (2011) consegue, para instâncias de tamanho pequeno a médio, gerar resultados muito melhores e em tempo computacional muito menor do que o algoritmo ORIGINAL, proposto por aquele autor. Para instâncias de tamanho grande, o POP-PPRDR conseguiu gerar iterações e melhorias que não tinham sido possíveis no algoritmo ORIGINAL. No próximo capítulo são sugeridos estudos de diversas naturezas, objetivando maximizar a eficiência do algoritmo proposto.

5. Conclusões e Trabalhos Futuros

O objetivo deste trabalho é propor uma técnica capaz de prover solução para o PPRDR em RDCs de médio e grande portes. Neste capítulo são apresentadas algumas conclusões de acordo com o observado nos resultados dos testes realizados, bem como algumas sugestões de trabalhos futuros.

5.1 Conclusões

O trabalho de NEVES (2011) examina diversas abordagens para o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR) – uma variante do Problema de Posicionamento de Réplicas (PPR) – que é um dos problemas que ocorrem em Redes de Distribuição de Conteúdos (RDCs). Os experimentos computacionais mostram resultados de boa qualidade, quando comparados a resultados existentes na literatura. No entanto, não se consegue obter resultados significativos para a versão *off-line* do Problemas de Posicionamento de Réplicas e de Distribuição de Requisições, PPRDR, em instâncias com mais do que 100 servidores. O autor cita, como um possível trabalho futuro, o estudo de uma metaheurística diferente daquela empregada em seu trabalho para tratar tais instâncias.

O presente trabalho empregou o *framework* POPMUSIC (*Partial OPtimization Method Under Special Intensification Conditions*), especialmente eficiente para resolver problemas combinatórios difíceis e de larga escala que possam ser otimizados por partes (TAILLARD e VOSS, 2001), para montar um algoritmo que visa obter uma solução

para o PPRDR em RDCs de médio e grande portes. A partir de uma dada solução inicial, não necessariamente boa, POPMUSIC orquestra um processo que inclui a definição de seus componentes livres: o que representa uma parte da solução; a expressão de uma função de relacionamento entre as partes que oriente a formação de subproblemas com partes fortemente relacionadas; a decisão do procedimento de seleção de uma parte como semente para formar um subproblema; e a definição do processo de otimização ao qual cada subproblema será submetido. O algoritmo trata da incorporação das melhorias eventualmente obtidas pelo processo de otimização de cada subproblema à solução completa e recomeça o processo montando novo subproblema e submetendo-o ao processo de otimização, parando quando não há mais subproblemas a otimizar. Com as definições adequadas de partes e subproblemas, POPMUSIC garante que a cada melhoria obtida em um subproblema corresponda uma melhoria no problema completo. Dada uma solução inicial e escolhidos os componentes, a metaheurística se encarrega de criar e melhorar o resultado de subproblemas do PPRDR em instâncias de uma RDC.

Neste trabalho, a solução inicial foi fornecida pela HNH (*Hybrid Network Heuristic*). Uma parte foi definida como um servidor com seus conteúdos e parcelas de requisições que lhe foram atribuídas. O valor da função de relacionamento entre duas partes foi definida como aumentando com o inverso da distância entre seus respectivos servidores. As partes disponíveis para servir de semente para gerar subproblemas foram organizadas num conjunto *O* (começando com todas as partes da solução inicial), tratado como uma fila seguindo a estratégia FIFO (*First In, First Out*). A cada iteração a primeira parte na fila serviu de semente para montar um subproblema que foi então examinado. Quando não houve melhoria, a semente foi retirada de *O*. Quando houve, todas as partes do subproblema que não estavam em *O* foram ali reinseridas, depois do último elemento. O processo de otimização de cada subproblema, ILSAM, foi baseado nos métodos de NEVES (2011), modificando-se o critério de parada: enquanto no algoritmo original era tempo, neste trabalho limitou-se a quantidade de iterações sem melhora.

De acordo com HOOS e STÜTZLE (2005), praticamente todo problema envolvendo interação com o mundo real tem restrições de tempo e, idealmente, um algoritmo pensado para resolver um problema desse tipo deve oferecer soluções razoavelmente boas em qualquer momento durante sua execução. Em casos de

problemas de otimização, isso tipicamente significa que o tempo de execução e a qualidade da solução mantenham uma correlação positiva. No presente estudo, que trata um problema de otimização, os testes geraram resultados alinhados com essa recomendação.

Os resultados alcançados comprovam que a técnica empregada, POP-PPRDR, usando o POPMUSIC conjugado aos métodos de NEVES (2011), oferece um caminho viável para resolução do PPRDR em instâncias de tamanho médio e grande.

5.2 Trabalhos Futuros

Há casos na vida real em que a qualidade do resultado é mais importante do que o tempo de processamento, e vice-versa. Com a estratégia de implementação adotada nos testes realizados, verificou-se que um percentual significativo (75 a 100%) da melhora obtida ocorre após uma quantidade de iterações menor ou igual a duas vezes a quantidade de partes da instância examinada. Em alguns casos, quando o percentual de 100% não foi atingido até tal marca, o algoritmo precisou dobrar a quantidade de iterações (e o tempo) para alcançar a melhoria restante.

Todos os testes relatados no presente trabalho utilizaram um mesmo valor para *r* (6), o tamanho dos subproblemas. A aplicação de valores menores para o parâmetro *r* fez o POP-PPRDR gerar resultados em tempos significativamente menores (tanto menor quanto menor o *r*). O aumento de *r* gera subproblemas de tamanho maior, que tendem a exigir mais tempo para solução pelo ILSAM. No caso de melhora obtida no resultado de um subproblema, isso costuma resultar numa quantidade maior de partes reabilitadas a servir de semente, o que significa maior número de iterações até convergir para o final do algoritmo. Uma quantidade maior de iterações, sendo cada uma delas mais longa, se traduz num aumento na duração total do algoritmo. Em termos dos valores alcançados para a função objetivo, não houve uma tendência clara; os valores ora melhoraram, ora pioraram com o aumento de *r*. Não foi possível obter um mesmo valor de *r* que gerasse o melhor resultado para todos os tamanhos de instância testados. Pode ser interessante ampliar o estudo da influência do valor de *r* nos resultados e tempos alcançados, não buscando um valor fixo mas uma regra como por exemplo, variar o valor de *r* em função do número $|V|$ de servidores. Uma alternativa que parece promissora é fazer *r* = maior valor inteiro contido na raiz quadrada de $|V|$.

Outro estudo que pode merecer atenção é avaliar se, e como, haveria uma correlação entre a dispersão geográfica dos servidores em uma RDC e o tamanho dos subproblemas que trouxesse melhores resultados na solução de um problema.

Uma possível variação na montagem dos subproblemas poderia incluir a escolha de uma parte com fraco grau de relacionamento com a semente, forçando uma diversificação.

Também parece válido testar estratégias de implementação do algoritmo pelo POPMUSIC diferentes daquela utilizada neste trabalho, em especial quanto à seleção de sementes do conjunto de controle *O* (ver Seção 3.2.1).

Podem ainda ser propostos os seguintes trabalhos futuros:

- Estudar os efeitos de utilizar o modelo acelerado de convergência proposto por TAILLARD *et al.* (2006), retirando do conjunto de controle *O* todas as partes de um subproblema para o qual não se obtenha melhora, no lugar do modelo clássico adotado neste trabalho, no qual se retira de *O* apenas a parte semente;
- Ampliar os experimentos variando a quantidade máxima de iterações sem melhora que define o critério de parada da ILSAM para tentar obter um melhor balanceamento entre qualidade dos resultados e tempo para obtê-los. Os testes realizados indicam que a qualidade dos resultados melhora conforme se aumenta o limite de iterações sem melhora, antes de parar os *loops* da busca local. O percentual de melhora do resultado, nos testes em que se variou a quantidade máxima de iterações, foi proporcionalmente inferior ao aumento observado no tempo de execução;
- Pesquisar a adoção de critérios de parada diferentes do proposto pelo *template* POPMUSIC, para casos em que se prefira um possível sacrifício na qualidade dos resultados em troca de menores tempos de resposta. Pode-se pensar, por exemplo, em limitar o tempo ou a quantidade de iterações (esta última poderia ser estabelecida como função da quantidade de partes da instância). Por exemplo, uma observação dos resultados de POP-PPRDR nas proximidades da 100ª iteração (para instâncias de tamanho 100, ou seja, um número igual à quantidade de partes na solução inicial) apresenta uma melhora média 113 vezes maior do que a melhora média obtida pelo

algoritmo ORIGINAL, sem o emprego da metaheurística POPMUSIC, com um consumo de tempo de apenas 77,18% do tempo gasto por ORIGINAL. Esse estudo poderia ser relacionado ao da estratégia de implementação, citado anteriormente;

- Estudar o estabelecimento de um limite mínimo para passar a considerar melhoras pequenas como significativas, acelerando a parada do POPMUSIC. Verificou-se nos testes a ocorrência de melhoras equivalentes a menos do que 0,5% do valor de outras. Como o algoritmo, independentemente do valor da melhora, reabilita todas as partes de um subproblema melhorado que tenham sido retiradas de *O* para que possam voltar a servir de semente, o algoritmo é levado a prolongar sua execução, às vezes devido a melhorias ínfimas. Um possível modo de implementar esta ideia é registrar o valor das melhoras obtidas em determinado período e considerar como "insignificantes" os laços de busca local que produzam melhora menor do que certo percentual do valor médio das melhoras ou, mais agressivamente, menor do que certo percentual da melhor melhora obtida. As melhoras obtidas nos laços considerados insignificantes poderiam ou não ser incorporadas à solução.

Referências

AHUJA, R. K., MAGNANTI, T. L., ORLIN, J. B., 1993, *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

AIOFFI, W., MATEUS, G., ALMEIDA, J. *et al.*, 2005, "Dynamic content distribution for móbile enterprise networks", *IEEE Journal on Selected Areas in Communications*, v. 23, nº 10 (Oct), pp. 2022-2031.

ALMEIDA, J. M., EAGER, D. L., VERNON, M. K. *et al.*, 2004, "Minimizing Delivery Cost in Scalable Streaming Content Distribution Systems", *IEEE Transactions on Multimedia*, vol. 6, nº 2, pp. 356-359.

ALVIM, A. C. F., TAILLARD, É. D., 2009, "POPMUSIC for the point feature label placement problem", *European Journal of Operational Research*, nº 192, pp. 396-413.

ALVIM, A. C. F., TAILLARD, É. D., 2013, "POPMUSIC FOR THE WORLD LOCATION-ROUTING PROBLEM", *EURO Journal on Transportation and Logistics*, nº 2, pp. 231-254.

BAKIRAS, S., LOUKOPOULOS, T., 2005, "Combining replica placement and caching techniques in content distribution networks", *Computer Communications* 28, pp. 1062-1073.

BARTOLINI, N., LO PRESTI, F., PETRIOLI, C., 2003, "Optimal Dynamic Replica Placement in Content Delivery Networks". In: *Proceedings of 11th IEEE International Conference on Networks – ICON*, pp. 125-130, Sydney, Australia, Sep.

BEKTAS, T.; OGUZ, O.; OUVEYSI, I., 2007, "Designing cost-effective content distribution networks", *Computers & Operations Research*, nº 34, pp. 2436-2449. Elsevier.

BORTOLIN JÚNIOR, S., GIRARDI, A., 2005, *Teoria dos Grafos aplicada ao Campeonato Brasileiro 2003.* Urcamp. Infocamp, Alegrete, Brasil.

BRESLAU, L., CAO, P., FAN, L. *et al.*, 1999, "Web Caching and Zipf-like Distributions: Evidence and Implications". In: *Proceedings of 18th IEEE INFOCOM*, v. 1, pp. 126-134, New York, USA, Mar.

DORIGO, M., MANIEZZO, V., COLORNI, A., 1991, *Ant System: An autocatalytic Optimizing Process.* Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy.

GAREY, M. R., JOHNSON, D. S., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, USA.

GASPARY, L. P., 1996, *Multicast Tutorial e Multicast IP*. UFRGS, Porto Alegre, Brasil.

GENDREAU, M., POTVIN, J.-Y., (Eds.), 2010, *Handbook of Metaheuristics.* 2 ed. International Series in Operations Research & Management Science, vol. 146. Springer Science+Business Media.

GINJUPALLI, P., 2005, *Study on internet delays*. M.Sc. dissertation, Umea Universitet, Umea, Sweden.

GLOVER, F., 1986, "Future Paths for Integer Programming and Links to Artificial Intelligence", Computer & Operations Research, vol. 13, nº 5, pp. 533-549.

GLOVER, F., KOCHENBERGER, G., (Eds.), 2003, *Handbook of Metaheuristics.* 1 ed. Kluwer Academic Publishers.

HOLLAND, J. H., 1992, *Adaptation in Natural and Artificial Systems.* MIT Press.

HOOS, H. H., STÜTZLE, T., 2005, *Stochastic Local Search – Foundations and Applications*. 1 ed. San Francisco, Morgan Kaufmann Publishers.

HUANG, C., ABDELZAHER, T., 2004, "Towards Content Distribution Networks with Latency Guarantees". In: *Proceedings of 12th IEEE International Workshop on Quality of Service – IWQOS*, pp. 181-192, Montreal, Canada, Jun.

[26] M. Crovella and A. Bestavros, 1996, "Self-similarity in world wide web traffic: Evidence and possible causes". In: *Proceedings of SIGMETRICS' 96*, May.

[27] S. Saroiu, K. P., Gummadi, R., Dunn, S. D. *et al.*, "An analysis of Internet content delivery systems." In: *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, USA, Dec.

KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P., 1983, "Optimization by Simulated Annealing". In: *Science, New Series*, vol. 220, nº 4598, pp. 671-680.

LI, F., GOLDEN, B., WASIL, E., 2007, "A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem", *Computers & Operations Research*, nº 34, pp. 2734-2742.

MARTELLO, S., TOTH, P., 1990, *Knapsack Problems*. John Wiley and Sons, Chichester.

MERRIAM-WEBSTER Dictionary. http://www.merriam-webster.com/dictionary Acessado em 2013.

MLADENOVIC, N., HANSEN, P., 1997, "Variable Neighborhood Search", *Computer & Operations Research*, vol. 24, nº 11, pp. 1097-1100.

NEVES, T. A., UCHOA BARBOZA, E., DRUMMOND, L. M. A. *et al.*, 2008, "Otimização em Redes de Distribuição de Conteúdos". *LX Simpósio Brasileiro de Pesquisa Operacional – SBPO*.

NEVES, T. A., 2011, *Redes de Distribuição de Conteúdo: Abordagens Exatas, Heurísticas e Híbridas.* Tese de D.Sc., Universidade Federal Fluminense, Niterói, Brasil.

OSTERTAG, A., DOERNER, K. F., HARTL, R. F. *et al.*, 2009, "Popmusic for a real World Large Scale Vehicle Routing Problem with Time Windows", *Journal of the Operational Research Society*, vol. 60, nº 7, pp. 934-943.

REEVES, C. R.; BEASLEY, J. E., 1993, "Introduction". In: Reeves, C. R.. (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Introduction chapter, Blackwell Scientific Publications. Oxford.

SHAHABI, C., BANAEI-KASHANI, F., 2002, "Decentralized Resource Management for Distributed Continuous Media Server", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, nº 6.

SUBRAMANIAN, A., 2012, *Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems*. Tese de D.Sc., Universidade Federal Fluminense, Niterói, Brasil.

TAILLARD, É. D., VOSS, S., 2001, "POPMUSIC – Partial Optimization Metaheuristic Under Special Intensification Conditions". *Journal of Network and Computer Applications*, vol. 2, nº 30, pp. 515-540.

TAILLARD, É. D., WAELTI, P., ALVIM, A., 2006, *Méthode de Décomposition POPMUSIC.* Presentation slides, Haute École d'Ingénierie et de Gestion, Yverdon, Switzerland.

TAILLARD, É. D.; RAILEANU, L.: WAELTI, P., 2007, “Preprocessing and Clustering Large-scaled Problem Instances”. *Metaheuristic International Conference 07*. Presentation slides. Montreal, Canada.

TENZAKHTI, F., DAY, K., OULD-KHAOUA, M., 2004, “Replication algorithms for the world wide web”, *Journal of Systems Architecture*, 50, pp. 591-605.

VICARI, C., 2008, *Distributed Dynamic Replica Placement and Request Redirection in Content Delivery Networks*. Ph.D. thesis, Universitá Degli Studi di Roma “La Sapienza”, Rome, Italy.

WAUTERS, T., COPPENS, J., DE TURCK, F. *et al.*, 2006, “Replica Placement in Ring-based Content Delivery Networks”, *Computer Communications*, vol. 29, nº 16, pp. 3313-3326. Elsevier.

WAUTERS, T.; COPPENS, J.; DHOEDT, B. *et al.*, 2005, “Load balancing through efficient distributed content placement”. In: *Proceedings of Next Generation Internet Networks*, pp. 99-105.

WOLFSON, O.; JAJODIA, S.; HUANG, Y., 1997, “An adaptive data replication algorithm”, *ACM Transactions on Database Systems* 22, nº 2, pp. 255-314.

ZHOU, X., XU, C.-Z., 2007, “Efficient algorithms of video replication and placement on a cluster of streaming servers”, *Journal of Network and Computer Applications*, vol. 30, nº 2, pp. 515-540. Elsevier.