HEURÍSTICAS ILS PARA O PROBLEMA DE POSICIONAMENTO DE RÉPLICAS E DISTRIBUIÇÃO DE REQUISIÇÕES

Tiago Araújo Neves
Escola de Engenharia Industrial e Metalúrgica de Volta Redonda, UFF
Av. dos Trabalhadores 420, Volta Redonda, RJ, Brasil
tneves@id.uff.br

Luiz Satoru Ochi
Instituto de Computação, UFF
R. Passo da Pátria 156, bloco E, S. Domingos, Niterói, RJ, Brasil
satoru@ic.uff.br

Célio Albuquerque
Instituto de Computação, UFF
R. Passo da Pátria 156, bloco E, S. Domingos, Niterói, RJ, Brasil
celio@ic.uff.br

RESUMO

Uma rede de distribuição de conteúdos é uma rede sobreposta onde os servidores replicam conteúdos e distribuem as requisições dos clientes com o objetivo de reduzir o atraso, carga nos servidores e congestionamento da rede, melhorando assim a qualidade percebida pelos clientes. Contudo, devido às restrições nos servidores e custos envolvidos no processo de replicação, não é viável replicar todos os conteúdos em toda a rede. Neste trabalho, duas heurísticas ILS são propostas para resolver um problema dinâmico e *offline* ligado ao gerenciamento das RDC, chamado Problema de Posicionamento de Réplicas e Distribuição de Requisições. O objetivo geral é reduzir os custos operacionais sem violar as restrições dos servidores e tentar atender as expectativas dos clientes sempre que possível. Resultados são comparados com os obtidos por abordagens exatas e mostram que as heurísticas conseguem resultados melhores em instâncias sem ótimos provados.

Palavras Chave. Redes de Distribuição de Conteúdos, ILS, Problema de Posicionamento de Réplicas e Distribuição de Requisições.
Otimização Combinatória, Otimização Aplicada.

ABSTRACT

A Content Distribution Network is an overlay network where servers replicate contents and distribute clients' requests with the aim at reducing delay, server load and network congestion, hence improving the quality of service perceived by end clients. However, due to server constraints and costs involved in the replication process, it is not reasonable to replicate the contents over the entire network. In this work, two ILS heuristics are proposed to solve a offline dynamic problem related to CDN management, called the Replica Placement and Request Distribution Problem. The overall objective is to reduce the operational cost without violating servers constraints and satisfying clients' expectations whenever possible. The results are compared to the ones obtained by exact approaches and show that the heuristics overcome the exact methods in instance where the optimal values are not proved.

Keywords. Content Distribution Network, ILS, Replica Placement and Request Distribution Problem.
Combinatorial Optimization, Applied optimization.

1. Introdução

A demanda por conteúdos multimídia na Internet aumentou consideravelmente em todo o mundo. Em muitos casos, para satisfazer as expectativas dos clientes deste tipo de conteúdo é necessário algum tipo de suporte especializado, que ofereça alguma garantia de Qualidade de Serviço (*Quality of Service* - QoS) como atraso máximo e largura de banda mínima. Neste sentido, muitos estudos têm sido feitos na tentativa de encontrar maneiras de servir melhor a crescente demanda por estes conteúdos [Tenzakhti et al. (2004)].

Um modo de servir estes conteúdos de maneira eficiente é através do uso de Redes de Distribuição de Conteúdos (RDC) [Tenzakhti et al. (2004), Zhou and Xu (2007)], que são tipicamente redes sobrepostas [Kurose and Ross (2003)] usadas para posicionar réplicas dos conteúdos nas proximidades dos clientes reduzindo assim, o atraso, a carga nos servidores e o congestionamento da rede, possibilitando portanto melhorias na qualidade do serviço prestado. A demanda por este tipo serviço é real em vista que existem empresas especializadas em prover serviços de RDC para clientes, como Adobe e Fox Interactive, que querem ou precisam distribuir seus conteúdos [Neves (2011)].

Existem vários problemas de otimização dentro das arquiteturas de RDC. Entre eles estão o Problema de Localização de Servidores (PLS), o Problema de Replicação (PR), o Problema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR). Descrições para o o PLS e para o PR encontram-se em [Bektas et al. (2007), Zhou and Xu (2007)]. O PPR pertence à classe *NP*-difícil e consiste em encontrar os melhores locais (servidores) dentro da rede para colocar os conteúdos replicados de modo a reduzir os custos de transmissão [Aioffi et al. (2005)]. O PDR consiste em, dado um conjunto de servidores com réplicas posicionadas, encontrar a melhor distribuição das requisições entre os servidores de modo a reduzir os custos de transmissão.

O problema abordado neste trabalho, denominado *Problema de Posicionamento de Réplicas e Distribuição de Requisições* (PPRDR), é uma variante do PPR e consiste em encontrar os melhores locais para posicionar as réplicas dos conteúdos e definir quais servidores irão atender cada uma das requisições a fim de que as exigências de QoS das mesmas sejam satisfeitas sempre que possível e o tráfego dentro da rede seja minimizado.

Uma das contribuição deste artigo é tratar de maneira conjunta o PPR e o PDR para arquivos extensos, analisando não apenas os custos para associar as requisições aos servidores, mas também uma série de questões que raramente são abordadas de maneira conjunta, como banda mínima para os clientes, carga nos servidores e presença de múltiplos conteúdos. Uma série de outras questões, relacionadas ao fato dos conteúdos serem extensos, como possibilidade de uma requisição ser atendida por múltiplos servidores ao mesmo tempo e a possibilidade do atendimento a uma requisição se estender por vários períodos de tempo também são consideradas. Segundo o conhecimento dos autores, existem poucos trabalhos que tratam o PPRDR considerando tantas características reais simultaneamente Neves (2011)]. No entanto, existem vários trabalhos que tratam deste problema de maneira mais simplificada. Em [Zhou and Xu (2007)], por exemplo, o PR e o PPR são tratados de forma isolada. Neste trabalho requisitos de QoS não são considerados, os servidores são homogêneos e o conjunto de conteúdos é constante. Este é um dos poucos trabalhos que usam meta-heurísticas para resolução do problema. Em [Aioffi et al. (2005)] o PPR e o PDR são tratados de maneira conjunta, contudo, a QoS também não é considerada e as requisições só podem ser atendidas por um único servidor. Em [Bektas et al. (2007)] o PS, PPR e PDR são tratados em suas versões estáticas, onde apenas um período de tempo existe no horizonte de planejamento. Uma revisão bibliográfica mais detalhada pode ser encontrada em [Neves (2011)].

O PPRDR pertence à classe *NP*-Difícil e possuí grande aplicação prática. Para tratá-lo duas meta-heurísticas ILS (*Iterated Local Search*) são propostas para a sua versão *offline*: uma convencional [Glover and Kochenberger (2003)] e uma que usa de memória adaptativa, ambas propostas neste trabalho. O objetivo do uso destas heurísticas é verificar a qualidade das soluções apresentadas em [Neves et al. (2012)] onde são utilizadas duas estratégias exatas para o PPRDR mas nem todas as soluções são ótimos provados.

Apesar do uso de modelos dinâmicos *offline* não ser totalmente adequado para ambientes com alto grau de imprevisibilidade porque utilizam conhecimento da demanda futura e sabem, *a priori*, todas as mudanças que ocorrerão, tirando muitas vantagens disto, estes modelos são capazes de resolver bem os problemas e fornecem um bom parâmetro de comparação para os modelos *online*, que são mais adequados aos ambientes com alto grau de imprevisibilidade pelo fato de que os dados só são conhecidos com o passar do tempo. Apesar de ser um problema de natureza distribuída, as abordagens utilizadas neste trabalho são centralizadas e requerem a centralização dos dados, processo este que tem custo significativo especialmente para redes de grande porte. Contudo, como o estudo de algoritmos de centralização não são o foco deste trabalho optou-se por considerar que os dados já se encontram centralizados.

O PPRDR que considera de maneira conjunta o posicionamento das réplicas e distribuição eficiente das requisições, ainda é pouco explorado pela literatura apesar de sua visível importância econômica, fato este que justifica o desenvolvimento de técnicas eficientes para resolvê-lo. Além disso, outra grande motivação deste trabalho é a possibilidade de melhorar a qualidade e a eficiência dos serviços prestados pelos provedores de RDC. Estas duas metas vão ao encontro da demanda mundial por melhor aproveitamento de recursos vivenciada neste século, onde simplesmente apresentar soluções para os problemas não é mais suficiente. É necessário apresentar soluções de qualidade, que satisfaçam as necessidades dos clientes e que aproveitem de maneira racional os recursos disponíveis.

O restante do trabalho está organizado da seguinte maneira. Na Seção 2 é feita uma descrição PPRDR. A Seção 3 apresenta uma versão da meta-heurística ILS para o problema. Já na Seção 4 o algoritmo ILS com memória adaptativa é descrito. A Seção 5 expõe os resultados obtidos, bem como uma análise dos mesmos, e na Seção 6 encontram-se as conclusões do trabalho.

2. O Problema de Posicionamento de Réplicas e Distribuição de Requisições

O PPRDR consiste em encontrar o melhor posicionamento para as réplicas dentro da rede sobreposta e redistribuir as requisições entre os servidores, com o objetivo de reduzir a carga da rede sem violar as restrições de QoS das requisições. Este problema difere do PPR pelo fato de que as requisições são tratadas individualmente, e não de maneira aglomerada. Assim, requisições originadas de um mesmo ponto para um mesmo conteúdo podem ser tratadas por servidores diferentes caso isso seja vantajoso.

Para lidar com a natureza dinâmica do problema (surgimento de requisições, conteúdos, etc.) o horizonte de planejamento é discretizado em períodos, que tem duração, para efeito deste trabalho, de 60 segundos. Em cada período, podem surgir novas requisições e conteúdos. Alguns conteúdos podem ser removidos, algumas requisições podem deixar de existir, caracterizando o fim do atendimento, e as condições da rede, tais como RTT (*Round Trip Time*) e atrasos nos canais, podem mudar.

Uma requisição, neste trabalho, consiste em uma identificação, um conteúdo exigido e quatro informações relacionadas à QoS: atraso máximo, banda mínima, banda máxima e atraso local. Considere que cada cliente se conecta a um servidor da RDC (chamado

de servidor origem), faz uma requisição e, quando esta última for plenamente atendida, o cliente é desconectado. O atraso local mencionado anteriormente representa o atraso entre o computador do cliente e o servidor origem; O atraso máximo representa o valor máximo de atraso tolerado pelo cliente. As bandas mínima e máxima representam respectivamente o que o cliente deseja e qual a capacidade máxima deste cliente em termos de banda. Estes quatro elementos de QoS permitem que cada requisição seja tratada de modo diferenciado pela RDC. As requisições possuem demanda divisível, o que significa que uma requisição pode ser atendida por múltiplos servidores simultaneamente, que é uma abordagem comum em redes *Peer-to-Peer*. Cada conteúdo possui um tamanho e um servidor inicial (em que o conteúdo surge pela primeira vez). Os servidores possuem limites de banda e capacidades de disco heterogêneos. Devido à existência de custos associados com replicação dos conteúdos, a relação custo-benefício entre a redução de carga na rede e o custo de replicação deve ser analisada. Estes fatores tornam o posicionamento das réplicas um problema não trivial, no qual uma decisão precipitada pode acarretar grandes custos, inviabilidades nos requisitos de QoS ou ambos.

Para modelar a dinâmica de requisições considera-se que estas podem surgir e terminar dentro de um horizonte de planejamento. Uma requisição possui demanda por um conteúdo, mas esta demanda não é persistente. Uma vez que o conteúdo tenha sido obtido pelo cliente, a requisição é encerrada, o que caracteriza o fim do atendimento. Em ambientes de rede com QoS, uma qualidade mínima é exigida pelas requisições, contudo geralmente não há imposições sobre a qualidade máxima. Isto leva a um ponto chave da discussão sobre as demandas. Se o sistema da RDC for mantido de modo que todas as requisições tenham somente seus requisitos mínimos atendidos, ele (o sistema de RDC) pode estar sendo subutilizado. Garantir os requisitos mínimos não quer dizer que o sistema da RDC não pode oferecer mais qualidade aos seus usuários. Alguns servidores podem ter banda sobrando e capacidade para atender ainda melhor os clientes. Por exemplo, uma requisição que exige uma banda mínima de 5 Bps para um conteúdo de 100 B é atendida por um servidor que tem, no momento, 50 Bps de banda não utilizada. Se o sistema fosse mantido de modo que somente as exigências mínimas fossem atendidas, o servidor utilizaria apenas 5 kbps de sua banda, levando 20 segundos para finalizar o atendimento. Porém, se o servidor enviar os 50 Bps, apenas dois segundos seriam necessários. Contudo, para prover este tipo de atendimento otimizado é necessário que o servidor conheça o limite máximo da banda da requisição. Por exemplo, não faz sentido o servidor enviar o conteúdo utilizando os seus 50 Bps de banda excedente se o máximo que a conexão da requisição suporta é de 10 Bps. Considerar este tipo de atendimento otimizado torna o problema mais realista e abre amplas frentes de trabalho. Ao lidar com arquivos grandes, vídeos por exemplo, pode ser necessário mais de um período para atender uma requisição. Assim, o atendimento das demandas das requisições deve ser propagado ao longo do horizonte de planejamento modo que uma solução para o problema deve tentar atender o máximo possível destas demandas em cada período, minimizando assim o tempo de atendimento de cada requisição.

Assim sendo, as características do PPRDR abordado neste trabalho são: 1) Servidores Capacitados - existência de limites, heterogêneos, de banda e espaço em disco nos servidores; 2) Múltiplos Conteúdos - existência de mais de um conteúdo na rede; 3) Dinâmico - mudanças na rede, nos conteúdos e nas demandas ocorrem ao longo tempo; 4) *Offline* - mudanças que ocorrem ao longo do tempo são conhecidas *a priori*; 5) Atendimento por Múltiplos servidores - possibilidade de atendimento de uma demanda por mais de um servidor; 6) QoS - presença de requisitos de banda e atraso nas requisições e 7) Atendimento

maximizado - tentativa de atender melhor os clientes sempre que possível. Uma descrição mais detalhada do problema pode ser encontrada em [Neves (2011)].

3. Uma Heurística ILS para o PPRDR

A versão *offline* do PPRDR é ainda pouco explorada pela literatura. Deste modo, uma versão da meta-heurística *Iterated Local Search* (*ILS*) [Glover and Kochenberger (2003), Talbi (2009)] é proposta nesta seção com o objetivo de servir como parâmetro de comparação para instâncias do PPRDR onde os métodos exatos não conseguem provar os ótimos em tempos computacionais razoáveis [Neves et al. (2012)]. Para a construção deste algoritmo o PPRDR foi particionado em dois subproblemas: o PPR e o PDR. Segundo [Neves (2011), Neves et al. (2012)] é possível encontrar uma distribuição ótima de requisições em tempo polinomial, desde que o posicionamento de réplicas esteja definido. Assim, optou-se por utilizar o ILS para determinar o posicionamento de réplicas e usar o algoritmo de fluxo em rede proposto em [Neves (2011)] para resolver a distribuição de requisições.

Algoritmo 1 ILS

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

O Algoritmo 1 mostra o pseudocódigo para um *ILS* geral. Primeiramente, uma solução inicial é gerada aleatoriamente ou utilizando um algoritmo construtivo. A solução inicial é então submetida a um processo de refinamento através do uso de um algoritmo de busca local. Em seguida são repetidos os seguintes passos até que um critério de parada seja atingido: i) perturba-se uma solução semente encontrada anteriormente gerando uma nova solução. ii) refina-se a solução perturbada através de um procedimento de busca local. iii) A solução gerada nesta iteração é submetida a um critério de aceitação, que verifica se a solução gerada nesta iteração é adequada para ser usada como semente em outras iterações.

Apesar de o histórico de busca estar presente nas etapas de perturbação e aceitação no algoritmo muitas vezes este histórico não é considerado devido à grande dificuldade em decidir quais atributos do histórico de busca utilizar. No entanto, o uso adequado de memória adaptativa em heurísticas tem se mostrado um caminho efetivo na resolução de problemas de otimização [Taillard et al. (2001)] e portanto ignorar este histórico pode levar à construção de algoritmos pouco efetivos. Neste sentido, também é proposta neste trabalho uma versão meta-heurística *ILS* que utiliza o histórico para tentar escapar de ótimos locais, apresentada na Seção 4.

No ILS proposto, a heurística *HNH* [Neves (2011)] é utilizada como método de geração da solução inicial, A estratégia elitista escolhida como critério de aceitação e o tempo total de processamento usado como critério de parada. A seguir, são discutidos os demais componentes utilizados no *ILS* tradicional.

3.1. Busca Local

Explorar a vizinhança de uma solução para o PPRDR não é uma tarefa trivial uma vez que o problema possui uma dimensão temporal. Para fazer mudanças no posicionamento das réplicas em um período t do horizonte de planejamento, é preciso recalcular os custos de replicação dos períodos $t-1$ e também os custos de replicação do período t, que podem aumentar ou diminuir de acordo com o posicionamento no período $t+1$. Isto ocorre porque os custos de criação de réplicas usadas em um dado período t não são pagos neste período, mas sim no período anterior, neste caso $t-1$. Além disso, uma vez que o posicionamento de réplicas é alterado em um período t também pode ser necessário recalcular a distribuição de requisições para t e todos os períodos seguintes, fato este que dificulta a exploração exaustiva das vizinhanças.

Como estratégia de busca local do ILS é utilizada a heurística *Record-to-Record* (*RTR*) [Talbi (2009)]. Este algoritmo percorre a vizinhança de uma solução semente aleatoriamente e aceita um vizinho pior como nova semente desde de que o valor da função objetivo deste vizinho seja menor (para problemas de minimização) que o *Recorde* acrescido de um *Desvio*. O *Recorde* representa o valor da função objetivo da melhor solução encontrada até o momento. O *Desvio* é um valor que representa a distância máxima tolerada entre a melhor solução encontrada e uma solução à ser usada como semente. O valor do *Desvio* influi diretamente na diversificação do algoritmo. Quanto maior este valor, maior é o número de soluções semente e, consequentemente, maior será a região do espaço de soluções explorada. O algoritmo 2 apresenta um possível pseudocódigo para a heurística *RTR*.

Algoritmo 2 RTR(Solução s, Desvio)

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

Para gerar o vizinho aleatório quatro estruturas de vizinhanças são utilizadas: *Inserção de conteúdo-IC*, *Remoção de conteúdo-RC*, *Realocação de conteúdo-RLC* e *Mudança por popularidade-MP*. Devido à restrição de espaço, não é possível descrever em detalhes as estruturas de vizinhança e portanto apenas uma breve descrição será feita aqui. Contudo, esta descrição mais detalhada pode ser encontrada em [Neves (2011)]. Na *IC*, um conteúdo é inserido no disco de um servidor em um período, todos escolhidos aleatoriamente. Na *RC* um conteúdo é removido do disco de um servidor em um período, todos escolhidos aleatoriamente. Na *RLC*, um conteúdo é removido do disco de um servidor em um período e inserido em outro servidor no mesmo período. Na *MP* o conteúdo de menor popularidade é substituído pelo conteúdo de maior popularidade. Para este fim, um servidor e um período de tempo são escolhidos aleatoriamente. Todas as estruturas de vizinhança possuem a mesma chance de serem escolhidas e um número máximo de iterações sem melhora, dado por $(|T| + |C| + |S|/10) \times 2$ para as instâncias de até 50 servidores e por $|T| + |C|$ para as instâncias de com 100 ou mais servidores, é utilizado como critério de parada. O valor do parâmetro desvio escolhido para os testes é 0.0001.

3.2. Estratégias de Perturbação

As perturbações possuem um papel crítico dentro do mecanismo de diversificação do *ILS*. Estas perturbações devem ser capazes de permitir que o algoritmo escape de ótimos locais evitando no entanto as desvantagens de um reconstrução total como por exemplo perda de informações relevantes que possam estar presentes na solução atual.

A primeira estratégia de perturbação é uma combinação de todas as estruturas de vizinhança usadas pelo algoritmo *RTR*. Nesta perturbação, um movimento de cada estrutura de vizinhança é feito na solução e a seguir os cálculos referentes aos custos de replicação e distribuição de requisições são refeitos. As estruturas de vizinhança são aplicadas na mesma ordem em que são descritas na Seção 3.1.

A segunda estratégia é a reconstrução aleatória do conjunto de réplicas de um servidor. Nesta perturbação, são selecionados aleatoriamente um período e um servidor. Todos os conteúdos do servidor escolhido são removidos e novos conteúdos são inseridos aleatoriamente. Um mecanismo de viabilização é utilizado quando necessário [Neves (2011)].

A terceira estratégia é baseada em uma perturbação exposta em [Glover and Kochenberger (2003)] que utiliza execuções rápidas de uma meta-heurística para gerar novas soluções. A perturbação proposta neste trabalho utiliza o algoritmo *RTR*. Para tornar a execução do *RTR* mais rápida, o número de iterações sem melhora utilizado neste caso é reduzido em 90%.

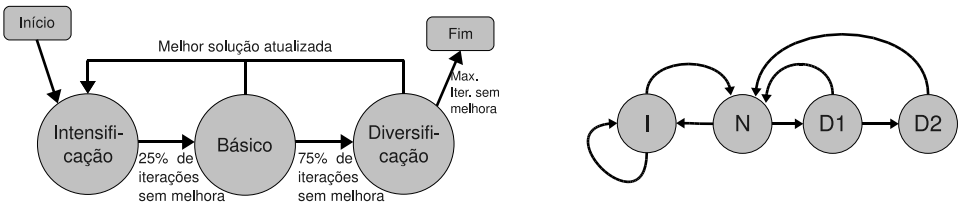
A última estratégia é a ressubmissão da solução incumbente à busca local. Normalmente, esta estratégia não é usada pelo fato de que a solução incumbente é um centro de atratividade e é resultado da submissão de outra solução à busca local. Isto quer dizer que, em muitos casos, a solução incumbente é um ótimo local e que simplesmente submeter esta solução à busca local novamente não traz benefícios. No entanto, como o *RTR* não explora as vizinhanças de maneira exaustiva é possível que uma ressubmissão da solução incumbente possa resultar em uma solução de melhor qualidade, visto que o algoritmo pode explorar uma parte da vizinhança não explorada anteriormente. Neste sentido, a ressubmissão é utilizada no *ILS* juntamente com as três primeiras estratégias de perturbação, sendo que, à cada iteração do *ILS*, uma destas quatro estratégias é escolhida aleatoriamente, tendo todas a mesma chance de escolha.

4. Um Algoritmo ILS com Memória Adaptativa

Segundo a literatura [Taillard et al. (2001)], o uso de memória adaptativa pode trazer melhorias na qualidade das soluções produzidas pelos algoritmos. Devido a este fato, é proposto nesta seção, um algoritmo *ILS* que utiliza estruturas simples de memória adaptativa. O algoritmo utiliza memória em dois componentes: dentro do *ILS*, através da alteração das chances de escolha das perturbações e alteração de parâmetros do *RTR*; E em uma versão do *RTR* que também utiliza memória para mudar o critério de aceitação deste algoritmo

O primeiro componente a ser descrito é uma versão adaptativa do *RTR*, chamada de *RTR-Adaptativo*. Este algoritmo alterna entre fases de Intensificação, comportamento Básico e Diversificação através de mudanças no parâmetro $Desvio$ e pode ter seu comportamento descrito pelo diagrama da Figura 1(a). É importante mencionar que apesar de o conceito de algoritmos adaptativos para diversos problemas de otimização não ser recente, este conceito ainda é pouco aplicado para o *RTR* [Neves (2011)].

Na fase de comportamento Básico o *RTR-Adaptativo* funciona exatamente como descrito na Seção 3.1. Na fase de Intensificação o valor do parâmetro $Desvio$ alterado para zero, o que faz com que apenas soluções melhores que a solução corrente sejam aceitas como novas sementes. Na fase de Diversificação o parâmetro $desvio$ tem seu valor aumentado em 10%, o que faz com que uma gama maior de soluções possa ser usada como novas



(a) Modelo do Algoritmo RTR-Adaptativo (b) Diagramas de Estados do *ILSAM*

Figura 1. Diagramas do algoritmo ILSAM

sementes. O algoritmo inicia na fase de Intensificação pelo fato de que o *RTR-Adaptativo* é usado como busca local, é razoável assumir que as soluções passadas como entrada devem ter sua vizinhança analisada de maneira mais criteriosa. Caso sejam atingidas 25% do número máximo de iterações sem melhora, o algoritmo passa para fase de comportamento Básico. Ao atingir 75% do número máximo de iterações sem melhora o algoritmo passa para a fase de Diversificação. Sempre que uma nova melhor solução é encontrada o número de iterações sem melhora é configurado para zero e o algoritmo volta à fase de Intensificação. As porcentagens de iterações usadas como transição de fases do algoritmo foram determinadas experimentalmente e todos os demais parâmetros de execução são os mesmos descritos na Seção 3.1.

Um fato importante a ser mencionado é que o mecanismo utilizado no *RTR - Adaptativo* não altera a complexidade das iterações. Segundo a literatura [Taillard et al. (2001)] em muitos casos o uso de memória adaptativa acaba por introduzir complexidade adicional às iterações. Entretanto, como as iterações dos algoritmos com memória adaptativa tendem a fornecer melhores resultados, o aumento na complexidade tende a ser compensado com uma redução no número de iterações. No mecanismo adaptativo proposto, as iterações não sofrem alteração em sua complexidade visto que os mesmos passos são seguidos nos dois algoritmos. A diferença é o comportamento em relação à aceitação de novas sementes. Na fase de Intensificação o *RTR-Adaptativo* tende a ser elitista, fazendo com que apenas a solução incumbente seja usada para gerar vizinhos. Na fase de comportamento Básico, o algoritmo procede da mesma maneira que o *RTR* tradicional que permite que soluções piores que a incumbente, mas que estejam dentro de um certo limite de qualidade, sejam utilizadas na geração de novos vizinhos. Na fase de Diversificação o valor do limite de qualidade é aumentado, fazendo com que uma maior gama de soluções possa ser usada como semente, aumentando a diversidade de soluções analisadas pelo algoritmo. É importante relatar que o mecanismo de memória proposto é utilizado apenas quando o *RTR* é usado como busca local. Portanto, nenhuma alteração no comportamento do *RTR* é feita quando este é usado como estratégia de perturbação.

A segunda estratégia de memória utilizada no ILSAM é a de alterar a probabilidade de escolha das perturbações e alterar parâmetros do *RTR*. A Figura 1(b) mostra o diagrama de estados do *ILSAM*. O círculo com a letra I representa o estado de intensificação. O círculo com a letra N representa o estado normal e os dois círculos com a letra D representam os estados de diversificação. Note que toda vez que o algoritmo for alternar entre intensificação e diversificação é necessário passar pelo estado normal.

O estado normal é o estado inicial do *ILSAM* e nele todas as perturbações possuem a mesma chance de escolha. Neste estado, há duas possibilidades de mudança: uma para o estado I e uma para o estado $D1$. Toda vez que o número de iterações executadas

é um múltiplo de 10, o algoritmo verifica a diferença entre a iteração atual e a iteração em que a solução incumbente foi encontrada. Caso esta diferença seja maior que 10 o algoritmo muda para o estado $D1$. Caso contrário, o algoritmo muda para o estado I e as probabilidades de escolha das perturbações são atualizadas.

No estado de intensificação a escolha das perturbações é feita através do mecanismo da roleta. Neste mecanismo cada um dos candidato recebe uma quantidade de fichas e estas fichas mudam de mãos de acordo com critérios estabelecidos. A chance de escolha de um candidato é proporcional à quantidade de fichas que ele possui. Neste trabalho, cada uma das perturbações recebe inicialmente 25 fichas. Em todos os estados do algoritmo são computadas estatísticas sobre a quantidade total de tentativas, número total de tentativas antes da última transição de estado e número de vezes em que a solução incumbente foi atualizada para cada uma das perturbações. No estado de intensificação estas estatísticas são utilizadas para fazer a troca de fichas, de modo que as perturbações de melhor desempenho tenham suas chances de escolha aumentadas. Mais detalhes podem ser vistos em [Neves (2011)]. Após a troca de fichas, todas as estatísticas referentes à última mudança de estados são reiniciadas. Isto é feito para que o algoritmo possa ter uma noção mais precisa do impacto das últimas alterações em seu comportamento. O número mínimo de fichas para cada uma das perturbações é 3, fazendo com que mesmo perturbações pouco eficientes tenha chance de escolha. Para verificar a transição de estado I, toda a vez que a iteração atual um múltiplo de 10 é verificado se a solução incumbente foi atualizada nas últimas 10 iterações. Caso afirmativo, o algoritmo permanece no estado I, fazendo as atualizações necessárias nas chances de escolha das perturbações. Caso contrário, o algoritmo passa para o estado N, onde todas as perturbações possuem a mesma chance de escolha.

No estado $D1$ todas as perturbações possuem a mesma chance de escolha e o valor do parâmetro $Desvio$ do algoritmo *RTR* é aumentado em 25%. Cabe lembrar que dentro do algoritmo *RTR-Adaptativo* o valor do $Desvio$ também é alterado. Isto permite que o *RTR-Adaptativo* possa analisar uma vizinhança ainda maior, visto que, o valor de $Desvio$, aumentado em 25% pelo *ILSAM*, é aumentado novamente em 10% no *RTR-Adaptativo*. Caso ocorram 10 iterações sem melhora algoritmo muda para o estado $D2$. Caso solução incumbente seja atualizada o algoritmo retorna ao estado N. O estado $D2$ é idêntico ao estado $D1$, porém o valor de $Desvio$ é aumentado 50%. Caso o algoritmo esteja no estado $D2$ e transcorram 10 iterações sem melhora, o algoritmo volta ao estado N. Todos os demais componentes do *ILSAM* são os mesmos utilizados pelo *ILS* descritos na Seção 3.

5. Resultados Computacionais

Os algoritmos foram implementados em C++ usando g++ versão 4.3 e executados em um Quad-Core com 2.83 GHz/core, 8 Gigabytes de RAM usando Linux. Para fins de comparação são usados os melhores resultados apresentados em [Neves et al. (2012)], que são obtidos por dois métodos exatos. Como estes métodos exatos apresentaram dificuldade na resolução apenas para as classes C e D de instâncias, somente as instâncias destas duas classes são usadas neste trabalho. Mais detalhes sobre as instâncias podem ser encontrados em [Neves (2011)].

A Tabela 1 mostra o comparativo dos *gaps* dos dois algoritmos. A primeira coluna de cada tabela indica as instâncias abordadas. As colunas 2 e 3 mostram, respectivamente, o melhor *gap* e o *gap* médio obtidos pelo *ILS* usando um limite de tempo como critério de parada (duas horas). Já as colunas 4 e 5 apresentam o melhor *gap* e o *gap* médio obtidos pelo *ILSAM*. Em negrito estão os melhores *gaps* encontrados. Com os *gap* médios são menores que 3%, pode-se dizer que as meta-heurísticas possuem um bom desempenho. É

importante notar que em diversos casos, marcados com "*", o resultado médio apresentado pelo *ILSAM* possui qualidade superior ao melhor resultado encontrado pelo *ILS*. O ILS obtem o melhor resultado em apenas em 6 instâncias. Em termos de resultados médios, o *ILSAM* é superior na grande maioria dos casos. Destaque deve ser dado para a instância 50D16 na qual os dois algoritmos foram capazes de encontrar soluções melhores que a encontrada pelos métodos exatos (*gaps* negativos). Isto se deve ao fato de que nem todas as soluções apresentadas em [Neves et al. (2012)] são ótimos provados.

(a) Instâncias 10/20 servidores

	Gap ILS (%)		Gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
10C11	0.08	0.12	**0.07**	0.10
10C12*	0.07	0.09	**0.06**	0.07
10C13	**0.15**	0.23	0.18	0.22
10C14*	0.40	0.44	**0.28**	0.36
10C15	0.36	0.46	**0.26**	0.37
10D16*	2.27	2.29	**1.93**	2.09
10D17*	2.00	2.05	**1.72**	1.82
10D18*	1.96	1.98	**1.61**	1.69
10D19*	1.54	1.59	**1.29**	1.39
10D20*	1.75	1.88	**1.52**	1.65
20C11*	2.65	2.69	**2.18**	2.27
20C12	**0.27**	0.31	0.29	0.3
20C13*	1.05	1.06	**0.97**	1.01
20C14	**1.10**	1.14	1.13	1.17
20C15	**0.34**	0.39	0.37	0.38
20D16*	1.96	2.03	**1.88**	1.89
20D17*	1.50	1.54	**1.44**	1.46
20D18*	2.18	2.2	**1.99**	2.02
20D19	2.47	2.48	**2.29**	2.82
20D20*	1.72	1.74	**1.64**	1.67

(b) Instâncias de 30/50 servidores

	Gap ILS (%)		Gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
30C11*	0.59	0.60	**0.55**	0.58
30C12*	0.19	0.20	**0.18**	0.19
30C13	**0.27**	0.28	0.28	0.29
30C14*	0.25	0.26	**0.23**	0.24
30C15*	0.18	0.18	**0.17**	0.18
30D16*	1.69	1.71	**1.5**	1.58
30D17*	2.58	2.61	**2.36**	2.41
30D18*	1.87	1.90	**1.78**	1.79
30D19*	1.20	1.23	**1.17**	1.19
30D20*	1.78	1.79	**1.61**	1.63
50C11*	0.342	0.344	**0.334**	0.341
50C12*	0.125	0.125	**0.124**	0.125
50C13	1.886	1.898	**1.881**	1.895
50C14*	1.978	1.981	**1.970**	1.977
50C15	**0.605**	0.609	0.609	0.611
50D16	-0.035	-0.025	**-0.043**	-0.032
50D17*	1.135	1.140	**1.119**	1.132
50D18*	1.806	1.819	**1.688**	1.734
50D19*	2.685	2.685	**2.679**	2.681
50D20*	1.081	1.095	**1.028**	1.051

Tabela 1. Gaps: ILS X ILSAM

A Tabela 2 mostra os resultados para as instâncias com mais de cem servidores. Nas instâncias da classe D o *ILSAM* obtém as melhores soluções e as melhores médias em todas as instâncias. As instâncias marcadas com "*" são instâncias onde o *ILSAM* apresenta resultados médios superiores à melhor solução do *ILS*. As instâncias marcadas com "+" são instâncias onde o *ILS* atinge resultados médios superiores à melhor solução do *ILSAM*. É importante enfatizar que ambos os algoritmos conseguem encontrar resultados melhores que os métodos exatos em vários casos.

Apesar dos algoritmos conseguirem resolver as instâncias de 200 a 400 servidores, estes resultados podem ser considerados pouco conclusivos visto que o número de iterações médio dos algoritmos é demasiadamente baixo para estas instâncias (duas iterações para as instâncias de 200 e uma iteração para as instâncias de 300 e 400) [Neves (2011)]. Para as instâncias com até 100 servidores pode-se concluir que o *ILSAM* apresenta resultados melhores visto que em grande parte dos casos o *ILSAM* apresenta resultados médios com qualidade superior aos melhores resultados obtidos pelo *ILS*, e também que na grande maioria dos casos o *ILSAM* obteve resultados médios melhores. Já nas instâncias com 200 ou mais servidores o número de iterações dos dois algoritmos é extremamente reduzido, o que por sua vez pode ocasionar resultados como observado para a instância 300C02. Neste caso, o algoritmo *ILSAM* não consegue atingir bons resultados, devido ao fato de que este algoritmo precisa de um número razoável de iterações para que os mecanismos de intensificação e diversificação possam ser efetivos.

Para determinar qual dos dois algoritmos é mais eficiente em termos de tempo computacional foi utilizada a metodologia *ttt-plot*. Esta metodologia consiste em analisar a probabilidade empírica de um ou mais algoritmos atingirem um alvo em um determinado

(a) Instâncias 100/200 servidores

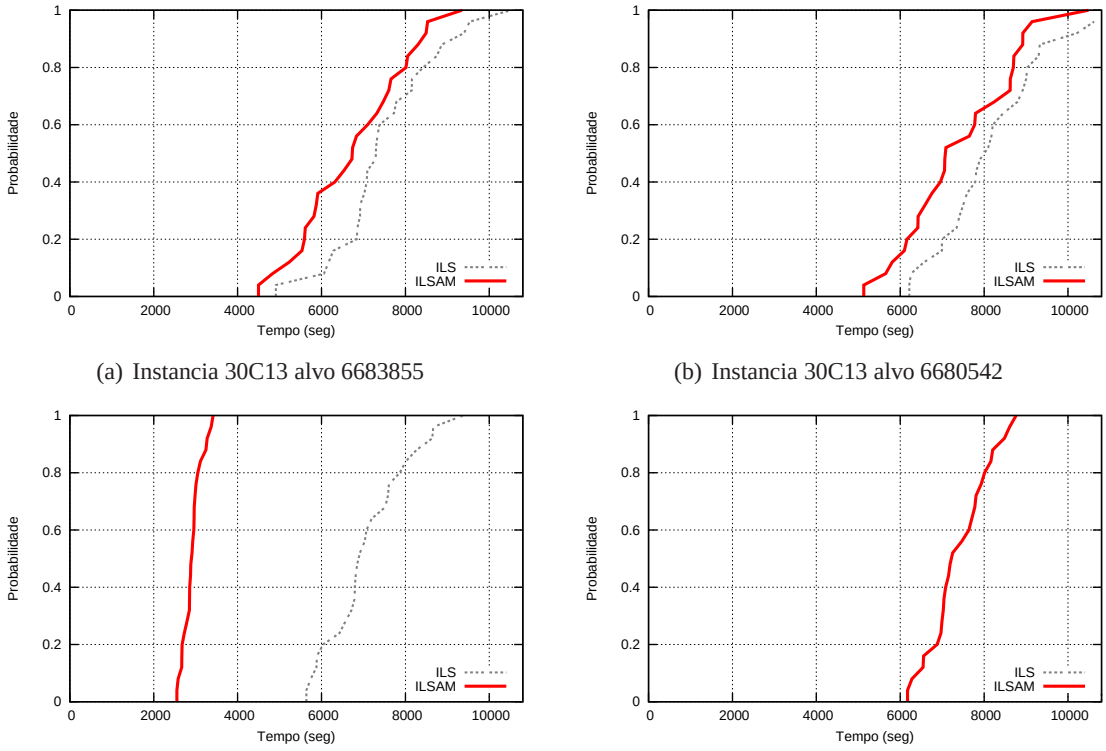
	Gap ILS (%)		Gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
100C01	**-0.0011**	-0.0004	**-0.0011**	-0.0004
100C02	**0.0000**	0.0000	**0.0000**	0.0000
100C03	**-0.0004**	-0.0002	**-0.0004**	-0.0002
100C04*	0.0000	0.0000	**-0.0047**	-0.0016
100C05	**-0.0002**	-0.0001	**-0.0002**	-0.0001
100D16	-0.0019	-0.0014	**-0.0023**	-0.0018
100D17*	-0.00006	-0.00003	**-0.00022**	-0.00009
100D18*	-0.0003	-0.0001	**-0.0003**	-0.0003
100D19	-0.0020	-0.0009	**-0.0026**	-0.0017
100D20*	-0.00245	-0.00098	**-0.00377**	-0.00312
200C01	**0.000000**	0.000000	**0.000000**	0.000000
200C02*	0.000000	0.000000	**-0.000008**	-0.000003
200C03*	0.000000	0.000000	**-0.000024**	-0.000009
200C04	**0.000000**	0.000000	**0.000000**	0.000000
200C05	**-1.630154**	-1.629840	-1.629931	-1.629559

(b) Instâncias 300/400 servidores

	Gap ILS (%)		Gap ILSAM (%)	
Instância	Melhor	Média	Melhor	Média
300C01*	0.000000	0.000000	**-0.000004**	-0.000001
300C02+	**-2.467219**	-2.463886	-2.462569	-2.462404
300C03	**-3.797414**	-3.797348	-3.797374	-3.797291
300C04	**-6.571241**	-6.571063	-6.571118	-6.571061
300C05	**0.000000**	0.000000	**0.000000**	0.000000
400C01	**-2.4903**	-2.4717	-2.4902	-1.6601
400C02	**-0.000004**	-0.000001	**-0.000004**	-0.000003
400C03	-6.23728	-6.2372	**-6.23733**	-6.2373
400C04	-4.05106	-4.0410	**-4.05113**	-4.0411
400C05	**-9.5629**	-6.8032	-9.5628	-6.3740

Tabela 2. Gaps: ILS X ILSAM - Mais de 100 Servidores

tempo. Quanto mais a esquerda estiver a curva de um algoritmo, mais rapidamente ele atinge o alvo. Além disso, ao se analisar as curvas de múltiplos algoritmos em um mesmo instante de tempo, pode-se determinar qual destes algoritmos possui uma maior probabilidade de atingir o alvo.



(a) Instancia 30C13 alvo 6683855

(b) Instancia 30C13 alvo 6680542

(c) Instancia 30D16 alvo 29178645

(d) Instancia 30D16 alvo 28568597

Figura 2. Distribuição dos Tempos de Execução

A Figura 2(a) mostra a distribuição dos tempos de execução para a instância 30C13 com alvo fácil. Esta instância foi escolhida porque o *ILS* obteve melhores resultados para esta instância. Apesar do *ILS* ter obtido melhores resultados nos experimentos anteriores, o gráfico indica que o *ILSAM* atinge o alvo mais rápido. A Figura 2(b) mostra os resultados para a mesma instância, porém com alvo difícil. Mesmo para um alvo difícil, o *ILSAM*

tende a convergir mais rapidamente. A curva do algoritmo *ILS* não alcança a borda superior do gráfico porque este algoritmo não consegue atingir o alvo em algumas execuções.

A Figura 2(c) mostra a distribuição dos tempos para a instância 30D16 com alvo fácil. Esta instância foi escolhida porque o algoritmo *ILSAM* obtém melhores resultados nesta instância. Em nenhum momento a curva do *ILSAM* encontra-se à direita da curva do *ILS*, indicando que o *ILSAM* converge para o alvo muito mais rapido. A Figura 2(d) mostra os resultados para a mesma instância com alvo difícil. A curva do *ILS* não aparece no gráfico porque o *ILS* consegue atingir o alvo em nenhuma das execuções.

6. Conclusões

Este trabalho descreve o PPRDR e expõe duas versões da meta-heurística *ILS* para o problema, uma convencional e uma que faz uso de memória adaptativa. Os resultados obtidos pelos algoritmos são comparados com resultados de métodos exatos da literatura mostrando que ambos os algoritmos conseguem atingir resultados melhores que os métodos exatos em instâncias onde os ótimos não são provados. Uma comparação entre os dois algoritmos revela que o algoritmo com memória adaptativa tende a fornecer soluções de melhor qualidade. Os testes também mostram claramente que o *ILSAM* tende a convergir mais rapido e o *ILS* convencional nem sempre consegue atingir os alvos.

É importante mencionar que o modelo computacional tratado neste trabalho inclui uma série de aspectos que não são tratados na literatura de maneira simultânea, tais como dinâmica nas condições da rede, surgimento e remoção de conteúdos e QoS.

Como Trabalhos futuros pretende-se testar outras meta-heurísticas para o problema e também tentar aprimorar os resultas obtidos por métodos exatos em [Neves et al. (2012)].

Referências

Aioffi, W., Mateus, G., Almeida, J., and Loureiro, A. (2005). Dynamic content distribution for mobile enterprise networks. *IEEE Journal on Selected Areas in Communications*, 23(10):2022–2031.

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

Glover, F. and Kochenberger, G., editors (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Kurose, J. and Ross, K. (2003). *Computer Networking: a Top-Down Approach Featuring the Internet*. Addison-Wesley.

Neves, T. (2011). *Redes de Distribuição de Conteúdos: Abordagens Exatas, Heurísticas e Híbridas*. Tese de Doutorado, Universidade Federal Fluminense - UFF.

Neves, T., Ochi, L., and Albuquerque, C. (2012). Soluções exatas para o problema de replicação e distribuição de requisições em redes de distribuição de conteúdos. Em *Anais do 30º Simpósio Brasileiro de Redes de Computadores SBRC 2012. A ser publicado*.

Taillard, E., Gambardella, L., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operation Research*, 135:1–16.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons.

Tenzakhti, F., Day, K., and Ould-Khaoua, M. (2004). Replication algorithms for the world-wide web. *Journal of Systems Architecture*, 50:591–605.

Zhou, X. and Xu, C.-Z. (2007). Efficient algorithms of video replication and placement on a cluster of streaming servers. *Journal of Network and Comp. Applications*, 30:515–540.