ESTRATÉGIA PARALELA PARA UMA METAHEURÍSTICA HÍBRIDA USADA NA CONFIGURAÇÃO DE UM SERVIÇO DE DISTRIBUIÇÃO DE VÍDEO

Gilberto Farias de Sousa Filho
Universidade Federal da Paraíba
Departamento de Ciências Exatas
Rio Tinto, PB - Brasil
gilberto@dce.ufpb.br

Alexander Almeida Pinto
Universidade Federal da Paraíba
Departamento de Informática
João Pessoa, PB - Brasil
alex@lavid.ufpb.br

Lucídio dos Anjos Formiga Cabral
Universidade Federal da Paraíba
Departamento de Informática
João Pessoa, PB - Brasil
lucidio@di.ufpb.br

Resumo. Este trabalho dá ênfase a implementação de uma estratégia paralela para uma metaheurística híbrida proposta para a configuração de um serviço utilizado para a distribuição de vídeo. Mais precisamente, esta metaheurística auxilia o serviço de distribuição de vídeo, denominado DynaVideo, em uma rede digital. Este serviço configura a rede de acordo com a variação da demanda. Uma variação na demanda é causada pelo número e localização dos clientes na rede. O problema resultante desta configuração do serviço pode ser modelado como um Problema da Árvore de Steiner. Alguns experimentos computacionais foram realizados com instâncias encontradas na literatura. Os resultados obtidos indicam que a nova estratégia abordada possui uma redução eficiente no tempo de processamento e na melhora dos resultados obtidos tanto comparados a metaheurística seqüencial quanto as estratégias paralelas anteriores.

Palavras-chave: Problema da Árvore de Steiner, Metaheurísticas, Programação Paralela.

Abstract. This work gives emphasis to implement a strategy for a parallel hybrid metaheuristic for configuration of a service used in the video distribution. In particular, we study the video distribution service called DynaVideo, on the digital network. This service uses mobile replication to adjust its configuration to demand variations, characterized by its clients number and placing. This problem can be formulated as Steiner Tree Problem. In following, we describe some computational experiments compared with the literature. The results indicate that the new strategy discussed is an efficient reduction in processing time and improved results compared to both sequential metaheuristic as the previous parallel strategies.

Key words: Steiner Tree Problem, Metaheuristics, Parallel Programming.

1. Introdução

Servidores que lidam com distribuição de vídeo, tais como a difusão de TV digital e transmissão de eventos ao vivo pela internet, normalmente trabalham com variações abruptas em sua demanda. O número, tipo e a localização dos clientes do serviço podem variar muito em um curto intervalo de tempo. Isso ocorre, por exemplo, toda vez que um programa de grande audiência começa a ser exibido.

Atualmente vários sistemas são capazes de distribuir vídeo em redes digitais. Esses sistemas são capazes de transmitir fluxos de vídeo em vários formatos. Entretanto, uma vez configurados, caso venha a ocorrer alguma variação na demanda durante a transmissão de um vídeo, os serviços de distribuição, que fazem uso de alguma plataforma, são incapazes de ajustar automaticamente a sua configuração dinamicamente.

Com o intuito de contornar a dificuldade exposta acima, foi projetada a plataforma Dynamic Video Distribution Service exposta por Leite (2001), para a distribuição de vídeos com suporte a variação de demanda. Isso é possível por que o serviço de distribuição de vídeo oferecido por esta plataforma foi concebido para distribuir vídeo de forma independente do seu formato e para interagir com diferentes tipos de clientes.

A principal característica do DynaVideo é a possibilidade de ajustar dinamicamente a sua configuração para uma dada demanda. O ajuste é feito com base no conceito de replicação móvel dos servidores de vídeo descrito por Kon (1999). Essas réplicas podem ser criadas em tempo real e movidas para pontos da rede que permitam o atendimento a uma determinada demanda.

A relevância deste trabalho deve-se a importância da reconfiguração dinâmica do serviço em um tempo computacional razoável através do paralelismo de um procedimento de otimização, onde todos os servidores ativos no momento cooperam de forma distribuída na computação da nova configuração. Assim, seguindo essa finalidade, este trabalho apresenta uma nova estratégia paralela de uma metaheurística híbrida GRASP descrita em Festa (2002), Aiex (2005) e ILS Lourenço et al. (2002), chamada GILS, para otimizar a reconfiguração do sistema DynaVideo, ou seja, sempre que o sistema detectar uma variação nas demandas, por exemplo, determinada pela entrada ou saída de algum cliente, a metaheurística será executada e, caso seja encontrada uma configuração melhor do que a atual, o serviço é reconfigurado automaticamente. Esta estratégia paralela se mostrou mais eficiente que as outras estratégias encontradas na literatura, tanto com relação às soluções encontradas quanto ao *speedups* de execução.

Este artigo está organizado em cinco seções. Os tópicos a serem cobertos em cada uma delas são descritos a seguir. Na seção 2, apresentamos uma breve descrição do sistema DynaVideo, o problema de reconfiguração dinâmica do serviço de distribuição de vídeo e a sua relação com o Problema da Árvore de Steiner. Ainda nesta seção, apresentamos alguns trabalhos existentes na literatura relacionados com a reconfiguração automática e a otimização na distribuição de vídeo. Na seção 3, descrevemos as estratégias propostas para a paralelização da metaheuristica híbrida encontrada na literatura e detalhamos a estratégia paralela definida neste trabalho. Na seção 4, apresentamos os resultados computacionais obtidos com o emprego desta metaheurística em algumas instâncias geradas de forma aleatória, para a versão seqüencial e suas paralelizações. Finalmente, na seção 5, são feitas algumas conclusões.

2. Arquitetura do Sistema DynaVideo

O serviço DynaVideo como descrito por Sousa (2005) é dinamicamente configurado. Esta flexibilidade permite que o serviço automaticamente se ajuste para variações em sua demanda, a idéia é que o serviço continuamente tente encontrar uma configuração otimizada para atender a uma dada demanda. No DynaVideo, a demanda é definida pelo número e localização dos clientes e dos servidores. A Figura 1 mostra os principais componentes do DynaVideo através de um diagrama de componentes.

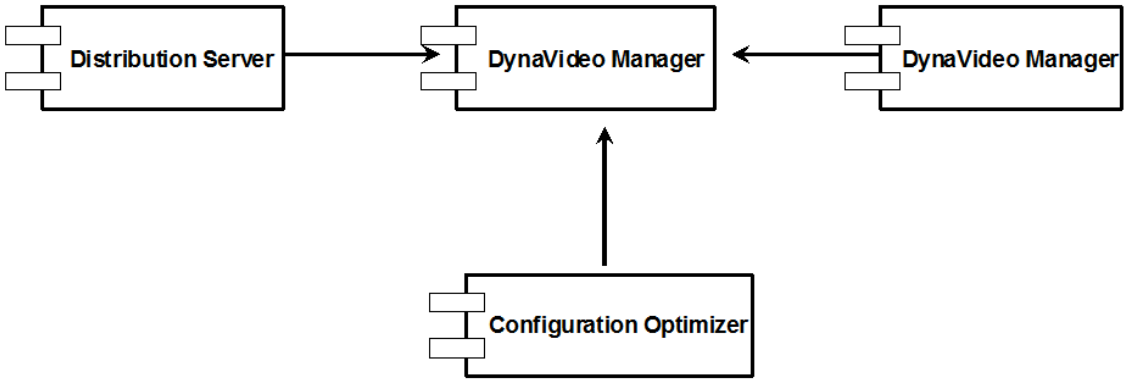

Figura 1. Principais componentes do DynaVideo.

O **DynaVideo Manager** (DM) controla a execução do serviço. Quando um cliente requisita um fluxo de vídeo, este módulo procura por um servidor com capacidade para atêndelo. Caso encontre, o DM associa o cliente a esse servidor. Caso contrário, um servidor contendo uma réplica será ativado para atender o mesmo.

O **Distribution Server** (DS) é o componente que trabalha de forma distribuida havendo uma instância em cada servidor e tendo acesso direto às fontes de vídeo, as quais podem ser codificadores em tempo real ou servidores de arquivo de vídeo. Quando o DS não faz acesso direto ao vídeo, ele trabalha como um refletor recebendo o fluxo de um servidor e repassando para os clientes a ele associados ou para outro servidor.

A chegada de um cliente determina uma mudança na configuração do serviço. Este evento ativa o **Traffic Monitor** (TM), descrito por Madruga (2000), para que este defina os servidores que se encontram mais próximos de cada, traçando as rotas dos servidores ativos até o cliente. Desta forma, o papel do TM é criar e atualizar uma estrutura de dados, o grafo de rotas, que armazena as rotas dos servidores ativos até os clientes.

O componente **Configuration Optimizer** (CO) foi inicialmente concebido para trabalhar de forma centralizada, veja Sousa (2005), porém com o paralelismo do algoritmo de otimização, este módulo passou a ter uma instância em cada servidor do sistema, sendo os mesmos ativados pelo DM para que esses se comuniquem e computem uma configuração ótima para o serviço. O CO é executado em **"background"**, procurando por uma melhor configuração para o serviço de distribuição, considerando a demanda corrente representada pelo grafo de rotas e o custo dos recursos de comunicação e processamento envolvidos na reconfiguração. Estando avaliada a nova configuração, o CO solicita ao DM que:

- Mova um cliente de um servidor para outro;
- Ative ou desative um servidor;
- Mova um servidor de um local para outro.

Otimização da Configuração do Serviço

No DynaVideo, a demanda é definida pelo número e localização dos clientes e servidores. Os custos envolvidos com o projeto da reconfiguração automática são:

- custo de ativação (ou configuração) dos servidores (*DS*);
- custo de ligação entre os servidores (*DS*);
- custo de atribuição (ou conexão) dos clientes aos servidores (*DS*).

O que efetivamente deseja-se otimizar é o tempo de atendimento ao cliente, em outras palavras maximizar a qualidade do serviço (QoS) através da reconfiguração do serviço. Os custos de ligação e atribuição podem ser avaliados, respectivamante, através de uma função que indica quantitativamente a qualidade da conexão entre os servidores e a qualidade da conexão com o cliente que está sendo atendido por um servidor. Essa função pode ter como parâmetros,

por exemplo, o número de *hops*, que dá uma idéia da distância entre o cliente e o servidor. O projeto da infra-estrutura do *backbone* deve atender aos requisitos dos clientes, a um custo mínimo. Para que possamos usar os três componentes de custo numa mesma função, optamos por estimar o atraso devido ao custo de ativação como um número adicional de hops, simulando um acréscimo no atraso do link.

O problema de encontrar uma árvore de custo mínimo, conectando um conjunto de nós, utilizando possivelmente nós auxiliares, é classicamente conhecido como *Problema da Árvore de Steiner* (*Steiner Tree Problem*), descrito por Winter (1997) . Assim, o problema de projeto da rede discutido acima é um caso particular do *Problema da Árvore de Steiner*, denominado *Steiner-Tree Star Problem (STSP),* apresentado por Lee (1996) e Xu(1996), visto que cada nó destino é conectado a apenas um nó ativo (servidor) numa topologia em estrela. Este problema foi abordado em Sousa (2005) usando a metaheurística GRASP e em Leite (2004) usando algoritmos genéticos e trangenéticos.

Em Sousa (2006) foram propostas três paralelizações da metaheurística híbrida GILS, detalhada na Figura 2, para a reconfiguração automática do serviço de distribuição de vídeo. O trabalho atual propõe uma quarta paralelização desta metaheurística híbrida que será apresentado na seção 3. Para verificar a eficiência da metaheurística híbrida usada, foram estabelecidos dois parâmetros: rapidez e qualidade da solução. No segundo parâmetro fizemos uso do modelo matemático apresentado por Lee (1996) e de um procedimento *branch-and-bound (B&B)*. Para medir a eficiência da estratégia paralela proposta, foram estabelecidos dois parâmetros: speedup e qualidade da solução, estes dois parâmetros foram comparados com o resultado obtido pela metaherística híbrida seqüencial.

3. Estratégias Paralelas

Nesta seção, serão levantadas as estratégias paralelas existente na literatura, além de propor uma nova estratégia paralela para o algoritmo híbrido seqüencial GILS, cujo código definido em Sousa(2006) é apresentado na Figura 2 e recebe os seguintes parâmetros: função de avaliação $f(.)$,função de construção gulosa $g(.)$, estrutura da vizinhança $N(.)$, número de movimentos para a perturbação $kmax$, e o percentual do tamanho da lista gulosa α. Na paralelização do algoritmo, o número de iterações GILS, dado por MAX_ITER_GILS, é distribuído dinamicamente entre os processadores disponíveis. Cada processador tem uma cópia do código do algoritmo GILS e mantém suas próprias estruturas de dados para armazenar a melhor solução encontrada, a solução corrente e a solução vizinha. Foram usadas diferentes sementes iniciais *RandomSeed* para a geração de números aleatórios em cada processador, de modo a evitar a repetição de seqüências aleatórias.

```
procedure GRASP+ILS(MAX_ITER_GILS, RandomSeed, f(.), g(.), N(.), kmax, α)
1.      f(s*) ← ∞;
2.      for i ← 1 to MAX_ITER_GILS do
3.              s ← ContrucaoGulosoAleatorio(g(.), α);
4.              s0 ← BuscaLocal(s);
5.              if (f(s0) < f(s*)) then
6.                      s* ← s0;
7.              end-if
8.              sILS ← ILS(f, N, kmax, s0, MaxIterILS);
9.              if (f(sILS) < f(s*)) then
10.                     s* ← sILS;
11.             end-if
12.     end-for
13.     return(s*).
end GRASP+GILS.
```

Figura 2. Algoritmo híbrido GILS (GRASP+ILS).

Em cada iteração do laço das linhas 2-12 é feita uma iteração GRASP seguido de uma iteração ILS. Para estes são utilizadas as vizinhanças ADD, DROP e SWAP. Na primeira um elemento é adicionado à solução (um servidor é ativo). Na segunda, um elemento é removido da solução (um servidor é desativado). Já na vizinhança SWAP ativa um novo servidor e desativa um servidor pertencente a solução em construção. Na fase de construção do GRASP apenas a vizinhança ADD é usada, enquanto que na fase de busca local todas as três vizinhanças são utilizadas de forma exaustiva. Para o algoritmo ILS é passada como solução inicial a melhor solução obtida pela busca local e são efetuadas kmax movimentos aleatórios do tipo ADD ou DROP, após estes movimentos o procedimento de busca local do algoritmo GRASP é acionado sobre a solução obtida.

3.1 Estratégias paralelas da literatura

Em Sousa (2006) foram definidas três estratégias de paralelização chamadas de PGILS1, PGILS2 e PGILS3. A estratégia de paralelização mais imediata para o algoritmo híbrido GILS, é obtida pela distribuição das iterações entre os processadores disponíveis. Tal estratégia é chamada PGILS1. Em um ambiente paralelo homogêneo, cada processador executa um número fixo de iterações GILS, igual ao número total de iterações MAX_ITER_GILS dividido pelo número de processadores. E uma vez que todos os processadores tenham terminado suas execuções, a melhor solução encontrada por eles pode ser obtida através de uma operação de redução.

A segunda estratégia de paralelismo se baseia na cooperação entre os processadores, em cada iteração, tão logo tenham executados os passos relativos à metaheurística GRASP, ou seja, após as suas respectivas fases de construção e busca local. O processador mestre efetua uma operação de redução para obter a melhor solução das execuções GRASP dos escravos e então a distribui aos escravos para que executem o algoritmo ILS a partir desta solução. Esta estratégia paralela denomina-se PGILS2.

A terceira estratégia busca minimizar a latência dos processadores escravos durante a comunicação com o mestre, presente na segunda estratégia, de modo que qualquer processador escravo ao enviar, para o processador mestre, a sua melhor solução da fase GRASP recebe imediatamente a melhor solução conhecida até o momento pelo mestre, determinada a partir das soluções recebidas de outros escravos. Assim sendo, processadores escravos podem estar em diferentes fases de execução do algoritmo, ou seja, um na fase GRASP e o outro já tendo se comunicado com o processador mestre e avançado à fase ILS. Também é possível que estejam em iterações distintas. Para que isto seja possível o processador mestre passa apenas a gerenciar a comunicação, ficando sempre apto a informar a melhor solução encontrada.

Todas as estratégias se baseiam no paradigma de troca de mensagens, num modelo abstrato de comunicação chamado mestre-escravo, ilustrado na Figura 3, onde só há troca de mensagens entre o processador mestre e um processador escravo.

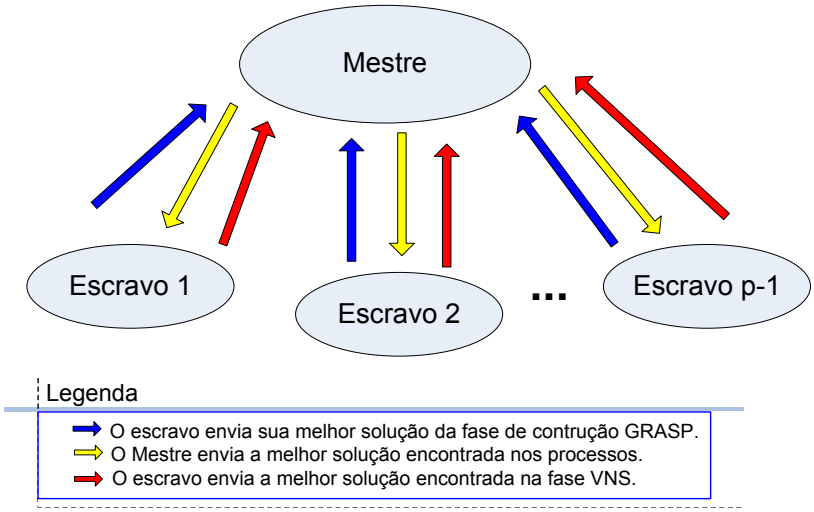

Figura 3. Troca de mensagens entre os escravos e o mestre.

Na segunda e terceira estratégias propostas, podemos verificar que apesar de haver cooperação entre os processadores a cada iteração GILS, a complexidade em termos do número de mensagens, ainda pode ser considerada baixa. Em cada iteração paralela do PGILS2 e do PGILS3, há apenas 3 mensagens trocadas entre cada escravo e o processador mestre, detalhadas na Figura 3, pelas legendas.

Na Figura 5 temos uma visão de como o processador mestre gerencia a comunicação, participando ativamente na execução das iterações GILS, na estratégia PGILS2. E na Figura 6 tem-se o processador mestre executando apenas o papel de gerente de comunicação, como detalhado na estratégia PGILS3. No caso da estratégia PGILS2 há um efeito não desejável no processador mestre, devido ao fato de que esse por ter mais atribuições que um processador escravo numa mesma iteração, termina por obrigar os processos escravos a terem uma certa latência a cada iteração, quando necessitam trocar mensagens com o mestre.

```
procedure PGILS2_Cliente(MAX_ITER_P, f(.), g(.), N(.), kmax, α)
1.      MPI_INIT();
2.      f(s*) ← ∞;
3.      for i ← 1 to MAX_ITER_P do
4.              s0 ← GRASP(s, g, α );
5.              Envia s0 para o processador mestre.
6.              Recebe smestre do processador mestre.
7.              sILS ← ILS(f, N, kmax, smestre, MaxIterILS);
8.              Enviar sILS para o mestre
9.      end-for
10.     MPI_FINALIZE();
end PGILS2_Cliente.
```

Figura 4. Algoritmo de um processador escravo na metaheuristica PGILS2.

De fato, os processos escravos só iniciam a fase seguinte de ILS após a operação de redução executada pelo processo mestre, como descrito na Figura 4.

```
procedure PGILS2_Mestre(MAX_ITER_P, f(.), g(.), N(.), kmax, α)
1.      MPI_INIT();
2.      f(s*) ← ∞;
3.      for i ← 1 to MAX_ITER_P do
4.              s0 ← GRASP(s, g, α);
5.              if (f(s0) < f(s*)) then
6.                      s* ← s0;
```

```
7.              end-if
8.              Sincroniza as soluções dos processos no mestre, de modo a que todos
                recebam a melhor solução da fase de construção desta iteração.
9.              sILS ← ILS(f, N, kmax, s0, MaxIterILS);
10.             O mestre recebe as soluções encontradas pelos outros processos e
                atualiza s*
11.     end-for
12.     return(s*).
13.     MPI_FINALIZE();
end  PGILS2_Mestre.
```

Figura 5. Algoritmo do processador mestre na metaheuristica PGILS2.

Em todas as estratégias propostas o ganho de aceleração obtido com a paralelização do algoritmo seqüencial é determinado pelo tempo de execução do processador escravo retardatário. Se as iterações não tiverem custos computacionais homogêneos, então isto pode se tornar uma dificuldade.

```
procedure PGILS3_Mestre(MAX_ITER_P, f(.))
1.      MPI_INIT();
2.      f(s*) ← ∞;
3.      for i ← 1 to (MAX_ITER_P * 2 * (size - 1)) do //size é o número de processos.
4.              Recebe qualquer mensagem de qualquer processo escravo.
5.              s ← solução do escravo;
6.              If( f(s*)> f(s)) then
7.                      s* = s;
8.              end-then
9.              If(TAG_MESSAGE == GRASP) then
10.                     Envia s* para o escravo
11.             end-then
12.     end-for
13.     return(s*).
14.     MPI_FINALIZE();
end PGILS3_Mestre.
```

Figura 6. Algoritmo do processador mestre na metaheuristica PGILS3.

3.2 Nova estratégia paralela proposta

Para superar a dificuldade gerada pelos processadores retardatários que atrasam o tempo de execução global das três estratégias anteriores, este trabalho propõe uma variação na estratégia PGILS3, tendo como objetivo diminuir o tempo de execução do algoritmo. A estratégia PGILS4 proposta, mantém a quantidade de execuções total do algoritmo, entretanto, cada escravo fica livre para executar quantas iterações for capaz, ou seja, diferente da PGILS3 onde cada escravo tem uma quantidade de iterações para processar fixa, o PGILS4 deixa este número livre, sendo responsabilidade do processo mestre contar quantas iterações globais foram processadas, independente de que escravo a gerou, e quando este número alcançar o número de iterações total do algoritmo, o mestre envia a seus escravos uma ordem de finalização do processamento. Esta idéia representa uma pequena modificação no pseudocódigo da estratégia PGILS3, entretanto, há uma alteração significativa no grau de granularidade do paralelismo, ou seja, tarefas menores são distribuídas ao longo das iterações, gerando menor latência para os processos escravos.

Essa estratégia é especialmente interessante quando o poder de processamento de cada nó é diferente, o que acontece nos servidores de vídeos presente na rede onde este desbalanceamento de carga é natural.

O algoritmo executado por cada escravo da Estratégia PGILS4 é idêntico ao algoritmo da estratégia PGILS2 ilustrado na Figura 2, havendo apenas uma alteração na linha 3, onde a condição de parada não é mais uma quantidade fixa de iterações, este loop passa a ser infinito, e todos os escravos na PGILS4 passam a ser interrompido pelo mestre através de uma mensagem de interrupção.

4. Resultados computacionais

As três estratégias propostas por Sousa (2006) e a estratégia paralela proposta neste trabalho para o algoritmo seqüencial híbrido GILS, descritas na seção anterior, foram desenvolvidas na linguagem C++ usando a biblioteca MPI (Otto, 1996) para implementar o mecanismo de troca de mensagens. Todos os experimentos computacionais foram feitos sobre um cluster montado em laboratório composto de 4 máquinas Intel Core 2 Quad, cada uma com a seguinte especificação: 4 processadores de 2.33 Ghz com 2 GB de memória RAM e rodando o sistema operacional sistema operacional Linux Ubuntu 9.10.

O número de iterações efetuadas pelo algoritmo GILS seqüencial foi dividido entre os processadores participantes de cada uma das execuções das diferentes estratégias paralelas, de modo a permitir o cálculo do ganho de aceleração do algoritmo, denominado *speedup*, que corresponde a razão entre o tempo de execução do algoritmo seqüencial e o tempo de execução do algoritmo paralelo para um particular número de processadores.

Para investigarmos o desempenho da metaheurística híbrida GILS e suas paralelizações, foram utilizadas 12 instâncias disponíveis na literatura (Sousa, 2005), com número de servidores variando em $N = \{10, 50, 100\}$ e número de clientes variando em $M = \{10, 50, 100, 1000\}$. Cada vértice (servidor ou cliente) teve suas coordenadas em $\Re^2$ geradas aleatoriamente com valores entre 0 e 100.

Para demonstrar a eficiência em termos de qualidade da solução da metaheurística GILS, realizamos comparações com um procedimento exato *B&B* (XPRESS MP, 2004), implementando o modelo STSP apresentado por Lee (1996). Para cada instância da Tabela 1 temos as duas primeiras colunas representando as dimensões das instâncias testadas e as colunas restantes divididas em dois grupos: procedimento *B&B* e *GILS*. No caso do procedimento *B&B*, a coluna z^* indica o valor ótimo e a coluna tempo indica o tempo computacional, em segundos, gasto na resolução da instância. Já para o grupo da metaheurística *GILS* as colunas adicionais, além da coluna tempo, são: *iter* que indica a iteração onde foi encontrada melhor solução, z que indica o valor obtido pelo *GILS* e, Δ (gap) que indica a diferença percentual entre as soluções:

$$\Delta = [\ (z - z^*) / z^*\ ] \text{ x } 100.$$

Instância		**Procedimento *B&B***		***GILS***			
n	**M**	z^*	**Tempo (s)**	**Iter**	Z	Δ (%)	**Tempo (s)**
10	10	587,083	75,5	2	587,083	0,00	0,09
10	50	1598,94	18,9	94	1662,319	3,96	0,136
10	100	2521,12	1,5	28	2593,732	2,88	0,197
10	1000	17878,9	3,6	37	17962,999	0,47	1,237

Tabela 1. Comparação entre o procedimento *B&B* e a metaheurística *GILS*.

Podemos observar, na Tabela 1, que a metaheurística *GILS* encontrou o valor ótimo para a instância $n = 10$ e $m = 10$. Nas demais instâncias, pode-se observar uma variação no valor do *gap*. Nesta variação, o maior valor obtido para o *gap* foi com a instância $n = 10$ e $m = 50$. Neste caso, obteve-se um *gap* igual a 3,96%. Verificou-se que o procedimento *B&B* obteve um tempo computacional superior a um minuto, mesmo para uma instância tão pequena. E para instâncias com n = 50, ou seja, para cinqüenta servidores o procedimento *B&B* não conseguiu alcançar uma solução ótima no limite de 6 horas de processamento. Os resultados para o procedimento *B&B* foram obtidos com o uso do software XPRESS MP Release 2003C, sob

licença acadêmica. Para o procedimento *B&B* não foram inseridos limites previamente calculados, de modo que a ferramenta utilizou apenas a relaxação linear.

Antes de apresentarmos os resultados obtidos com as estratégias paralelas, mencionaremos os valores adotados para os parâmetros da metaheurística *GILS*. Estabeleceu-se que o número máximo de iterações, denotado por MAX_ITER_GILS, fosse igual a 240, para cada estratégia e cada instância foram realizadas cinco execuções. O valor escolhido para α foi igual a 0.5, para qual se obteve o melhor desempenho durante os testes de parametrização do algoritmo seqüencial GILS, vide Sousa (2005).

Estratégia	**Processador**		
	4	**8**	**16**
PGILS1	1 %	0,98 %	0,89 %
PGILS2	1,1 %	0,48 %	0,64 %
PGILS3	-1,3%	-1,3%	-0,94%
PGILS4	**-1,5%**	**-1,2%**	**-1,1%**

Tabela 2. *Gap* médio das estratégias paralelas comparadas ao algoritmo seqüencial.

Para uma dada estratégia e um dado número de processadores é apresentado, na Tabela 2, o valor percentual de afastamento da média das soluções obtidas pela estratégia paralela com relação à melhor solução obtida pela estratégia seqüencial. Podemos observar que, em média, as estratégias PGILS3 e PGILES4 obtiveram melhores desempenhos, alcançando soluções melhores do que aquelas obtidas pelo algoritmo seqüencial GILS. A estratégia PGILS4 foi a estratégia que obteve a melhor solução média de todas as estratégias, chegando a melhorar em 1,5% o resultado encontrado pela metaheurística seqüencial. A estratégia PGILS2 teve o pior comportamento médio, demonstrando que a idéia de impor a todos os escravos a execução da fase ILS a partir de uma mesma solução, no caso a melhor encontrada por todos os escravos na fase GRASP, gerou baixa diversificação, incorrendo então em soluções de pior qualidade.

Problema		**GILS**	**PGILS1**		**PGILS2**		**PGILS3**		**PGILS4**	
n	**M**	Z	Z	Δ (%)	Z	Δ (%)	Z	Δ (%)	Z	Δ (%)
10	**10**	587,08	587,08	0,00	587,08	0,00	587,08	0,00	587,08	0,00
10	**50**	1662,32	1662,32	0,00	1662,32	0,00	1662,32	0,00	1662,32	0,00
10	**100**	2593,73	2593,73	0,00	2593,73	0,00	2593,73	0,00	2593,73	0,00
10	**1000**	17963	17963	0,00	17963	0,00	17963	0,00	17963	0,00
50	**10**	531,14	531,14	0,00	534,37	0,61	531,14	0,00	531,14	0,00
50	**50**	2166,16	2197,85	1,46	2183,74	0,81	**2129,58**	-1,69	2138,31	-1,29
50	**100**	2755,42	2757,65	0,08	2770,45	0,55	2721,04	-1,25	**2715,47**	-1,45
50	**1000**	10521,64	10489,63	-0,30	10488,82	-0,31	10443,05	-0,75	**10443,05**	-0,75
100	**10**	547,12	555,53	1,54	547,68	0,10	539,92	-1,32	**529,79**	-3,17
100	**50**	2045,37	2057,06	0,57	2082,38	1,81	1959,26	-4,21	**1938,94**	-5,20
100	**100**	3897,52	3931,87	0,88	3865,57	-0,82	3751,54	-3,75	**3730,98**	-4,27
100	**1000**	10161,52	10011,71	-1,47	10021,06	-1,38	9901,18	-2,56	**9893,68**	-2,64

Tabela 3. Comparação dos melhores resultados do GILS e as quatro estratégias paralelas.

A Tabela 3 apresenta uma comparação entre as melhores soluções encontradas pela metaheurística GILS e as estratégias paralelas do mesmo. Podemos observar nesta tabela que a estratégia PGILS4 melhora o melhor resultado conhecido em 6 instâncias das 12 testadas, sendo estas as instâncias de maior dimensão. Mostrando que o algoritmo se torna mais eficiente com o aumento do número de servidores.

Estratégia	Número de Processadores		
	4	8	16
PGILS1	4,03	7,87	15,35
PGILS2	3,89	7,21	13,54
PGILS3	3,06	7,10	14,51
PGILS4	3,11	7,10	14,78

Tabela 4. Média dos *speedups* por processadores para as estratégias paralelas 1, 2, 3 e 4.

A Tabela 4 mostra o *speedup* médio alcançado para as estratégias PGILS1, PGILS2, PGILS3 e PGILS4 onde a primeira estratégia, por incorrer em menor custo de comunicação, obteve *speedups* superiores. De fato, na estratégia PGILS1, há troca de mensagens apenas na operação de redução executada pelo processador mestre, ao final de sua execução, num custo que se torna quase desprezível à medida que se aumenta o número total de iterações do algoritmo paralelo. Daí a obtenção, em alguns casos, de *speedups* lineares.

As estratégias PGILS3 e PGILS4 também demonstraram boa escalabilidade, mesmo estas estratégias tendo um processador a menos trabalhando (o processador mestre apenas gerencia mensagens), seus *speedups* se aproximam da estratégia PGILS1. E para um número maior de processadores haverá uma diminuição do impacto sobre o *speedup*, como podemos observar no gráfico da Figura 7. Também podemos observar na Figura 7 que a estratégia PGILS4 proposta neste trabalho tem uma melhora no *speedup* com relação a estratégia PGILS3 quando o número de processadores aumenta.

5. Conclusões

A utilização do serviço de reconfiguração da rede de servidores de distribuição de vídeo, como no caso do DynaVideo, diminui o impacto do aumento da demanda sobre a rede, uma vez que esta reconfiguração reduz o volume total de tráfego decorrente do atendimento aos clientes. A replicação tende a aproximar um vídeo a uma dada área de clientes, onde este esteja tendo uma elevada demanda, colocando uma cópia do vídeo num servidor mais próximo destes clientes. Com o crescimento das aplicações de vídeo haverá certamente um incremento significativo do número de servidores de vídeos instalados, o que tornará vital o uso de uma estratégia paralela para a resolução do STSP. As paralelizações propostas para o algoritmo híbrido GILS, que combina as metaheurísticas GRASP e ILS, é empregado para ajustar dinamicamente a configuração do sistema face às modificações na demanda do serviço de vídeo.

O algoritmo seqüencial GILS teve bom desempenho, gerando boas soluções, no entanto, o tempo tende a crescer bastante num cenário realista, em uma rede que possua muitos servidores e clientes. Aliado a isto, há o caráter dinâmico de ajuste do serviço de vídeo, o que leva a se buscar estratégias de paralelização para o algoritmo GILS, de modo a reduzir o tempo computacional. Este trabalho propôs um novo esquema de paralelismo, que adota o modelo de comunicação mestre-escravo, onde o processo mestre não participa das iterações do algoritmo GILS. Para esta estratégia, chamada PGILS4, foi observado bom *speedus*, sendo este parâmetro melhorado com o aumento do número de processadores, o que mostra que esta estratégia que não fixa a quantidade de iterações a serem executadas por cada escravo, colocando o controle global das iterações do algoritmo para o mestre, tem boa escalabilidade. Na comparação com 16 processadores o PGILS4 perde apenas para a estratégia PGILS1, estratégia que possui um único momento de trocas de mensagem ao fim da execução global, lembrando que no PGILS4 o processador mestre não trabalha apenas gerencia seus escravos.

A primeira e segunda estratégia, PGILS1 e PGILS2, apresentaram soluções de pior qualidade do que as obtidas pelo algoritmo seqüencial, com *gap* médios de até 1,1%. Nas terceira PGILS3 e quarta PGILS4 estratégias, há um ganho nítido na qualidade das soluções devido à maior diversificação das soluções que são utilizadas durante o procedimento ILS, isto

ocorre porque o processador mestre, a cada iteração, responde ao processador escravo com a melhor solução obtida até então pelo outros escravos que já informaram, diferente da segunda estratégia quando o processador mestre ouve todos os processadores escravos e informa, de volta, a melhor solução encontrada dentre todos os escravos, ou seja, a mesma solução será passada ao procedimento ILS, que perturba aleatoriamente essa solução. No caso da estratégia PGILS4 que encontrou os melhores resultados, foram obtidas soluções com custo médio até 1,5% menores que o seqüencial.

A estratégia PGILS4 conseguiu melhorar o melhor resultado conhecido na literatura de seis das doze instâncias testadas, sendo estes problemas os de maior dimensão, mostrando que esta estratégia também melhora as soluções produzidas com o crescimento do problema.

Todas as estratégias paralelas propostas se mostraram eficientes, em termos de *speedup*, para as maiores instâncias testadas, o que sinaliza positivamente para a possibilidade de sua utilização prática, otimizando o serviço de distribuição de vídeo através da sua reconfiguração dinâmica e explorando o potencial de processamento distribuído da rede. Contudo para problemas maiores acredita-se numa vantagem da estratégia PGILS4 proposta neste trabalho, tanto em relação aos *speedups* alcançados, como em relação à qualidade das soluções.

Referências Bibliográficas

Aiex, R. M. e Resende, M.G.C. "*Parallel strategies for GRASP with path-relinking*", Metaheuristics: Progress as Real Problem Solvers, (T. Ibaraki, K. Nonobe and M. Yagiura, editores), pp. 301-331, Springer, 2005.

Festa, P. e Resende, M.G.C. "GRASP: an annotated bibliography", Essays and Surveys on Metaheuristics (C.C. Ribeiro e P. Hansen, editores), pp. 325-367, Kluwer Academic Publishers, 2002.

Kon, F.; Campbell, R. et al. "Dynamic Reconfiguration of Scalable Internet Systems with Mobile Agents", Technical Report, Department of Computer Science at the University of Illinois at Urbana-Champaign, 1999.

Lee, Y., Chiu S. Y. E Ryan J. A branch and cut algorithm for the Steiner tree-star problem", Informs Journal on Computing, 8(3):100-120, 1996.

Leite, L. E. C., de Souza Filho, G. e Batista, T. "DynaVideo - A Dynamic Video Distribution Service", 6th Eurographics Workshop on Multimedia, pp 95-106, 2001.

Leite, L. E. C., de Souza Filho, G.; Goldbarg, M. C. e Goldbarg, E. F. G. "Comparando algoritmos genéticos e transgenéticos para otimizar a configuração de um serviço de distribuição de vídeo baseado em replicação móvel", XXII Simpósio Brasileiro de Redes de Computadores, v. 1. pp. 129-132, 2004.

Lourenço, H. R., Martin, O. and Stutzle, T. (2002). Iterated local search, Handbook of Metaheuristics, Vol. 57 of Series in Operations Research & Management Science, pp. 321–353.

Madruga, Marcos; Batista, Thais; Lemos, Guido. "SMTA: Um Sistema para Monitoramento de Tráfego em Aplicações Multimídia". CLEI'2000. Cidade do México – México. September, 2000.

Otto, S. Huss-Lederman, S. Walker, D. Dongarra, J. Snir, M. Mpi: The Complete Reference (Scientific and Engineering Computation Series). Mit Pr, 1996

Sousa, Gilberto;Cabral, Lucídio; Macambira, Elder; Lemos, Guido. "Metaheurística Híbrida Paralela para a Configuração de um Serviço de Distribuição de Vídeo". XIII CLAIO. Montevideo, Uruguai. Setembro, 2006.

Sousa, Gilberto;Cabral, Lucídio; Macambira, Elder; Lemos, Guido. "Uma metaheuristica GRASP para configuração de um serviço de distribuição de vídeo baseado em replicação móvel". SOBRAPO'2005. Gramado-RS, Brasil. Setembro, 2005.

Sousa, Gilberto, Nery, L. D., Elias, G., Leite, L. E. C. e de Souza Filho, G: "Uma arquitetura hierárquica e distribuída para um serviço de distribuição de vídeo ao vivo", 10th Brazilian Symposium on Multimedia and the Web and 2nd Latin American Web Congress, v. 2, pp. 313-314, Ribeirão Preto-SP, Brazil, 2004.

Winter, P. e Zachariasen, M. "Euclidian Steiner Minimum Trees: An Improved Exact Algorithm", Networks, 30(3):149-166, 1997.

XPRESS MP. Getting staterd, Dash Optimization Ltd., 2004.

Xu, J.; Chiu, S. and Glover F. . Using Tabu Search to solve the Steiner tree-star problem in telecommunication network design. Telecommunications systems, Vol 6:117-125 (1996).