DEPARTAMENTO DE COMPUTAÇÃO

D E C O M

U F O P

UNIVERSIDADE FEDERAL DE OURO PRETO

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

"USO DE METAHEURÍSTICAS GRASP E BUSCA TABU NA RESOLUÇÃO DO PROBLEMA DE DIMENSIONAMENTO DE REDES IP"

FERNANDA SUMIKA HOJO DE SOUZA
VIVIANE DE SOUZA COELHO

Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti
Co-orientador: Prof. Dr. Marcone Jamilson Freitas Souza

Projeto de Iniciação Científica
Relatório Final para o CNPq

OURO PRETO - MG
AGOSTO/2004

SUMÁRIO

LISTA DE FIGURAS

RESUMO

O presente trabalho apresenta uma proposta de formulação e implementação de algoritmos baseados nas técnicas de otimização GRASP (*Greed Randomized Search Procedure*) e Busca Tabu para atender a nova geração da Internet, que implementa Qualidade de Serviço. Este novo contexto surgiu da crescente expansão da rede e da necessidade de atender a novos requisitos impostos por aplicações mais complexas, tais como transmissões em tempo real. São discutidos alguns conceitos relacionados à Qualidade de Serviço e Engenharia de Tráfego, além da proposta do projeto TEQUILA. Resultados computacionais são apresentados, comprovando que é possível prover uma melhora no dimensionamento da rede, através das técnicas propostas.

Palavras-chave: Qualidade de Serviço, Dimensionamento de redes, Engenharia de Tráfego, Metaheurísticas.

1 INTRODUÇÃO

A Internet, como foi proposta inicialmente, mostrou-se muito eficiente para atender aos requisitos do momento histórico em que fora criada, porém, não mais tem conseguido atender aos novos requisitos impostos por aplicações mais complexas que implementam troca de informações em tempo real, além do crescente aumento do número de usuários. Por isso, o atual desafio da Internet vem sendo alocar adequadamente os recursos da rede, para garantir um nível apropriado de qualidade de serviço e o correto funcionamento da rede, mesmo que esta esteja sob um alto nível de utilização.

Em resposta ao crescimento de demanda por qualidade de serviço na Internet, a Internet Engineering Task Force (IETF) estabeleceu dois grupos de trabalhos para desenvolver os Serviços Diferenciados (*DiffServ*) [Blake et al., 1998] e Serviços Integrados (*IntServ*) [Braden et al., 1994] com o objetivo de estender a arquitetura dos protocolos da Internet, comumente chamados TCP/IP em referência aos nomes dos seus dois principais protocolos, para prover qualidade de serviço.

Verificou-se que a questão de prover qualidade de serviço em uma rede IP está estritamente relacionada com a otimização do seu desempenho. Por exemplo, para atender aos requisitos de qualidade de serviço feitos por uma aplicação, poderia ser necessário redirecionar o tráfego não prioritário para outras rotas e redirecionar o tráfego prioritário para as rotas que melhor atendam as restrições e garantias exigidas pela aplicação. Na Internet, tal situação é denominada como uma operação de engenharia de tráfego. Awduche et al. [2002] definem o conceito de engenharia de tráfego como o aspecto da engenharia de rede que lida com a avaliação e otimização de desempenho de uma rede.

Para que os objetivos da engenharia de tráfego sejam atingidos na rede Internet, faz-se necessário definir caminhos explícitos entre os nós de uma rede de tal forma a garantir o mapeamento do tráfego nos diversos enlaces da rede. Na versão corrente da Internet, essa funcionalidade não é encontrada.

Para tal, foi idealizada a arquitetura MPLA (*Multiprotocol Label Architecture*) [Rosen & Callon, 2001] que permite que seja estabelecida uma rota explícita entre um nó de ingresso e um nó de egresso, denominada ER-LSP (*Explicitly Routed- Label Switched Path*) ou simplesmente LSP. Essa arquitetura define um esquema de encaminhamento de pacotes IP baseado em rótulos (labels), ao invés de endereços. A

implementação do esquema é feita através de um protocolo chamado MPLS (*Multiprotocol Label Switching*) [Rosen & Callon, 2001].

Para que rotas explícitas sejam convenientemente definidas e que atendam aos objetivos da engenharia de tráfego e satisfaçam as restrições impostas, técnicas baseadas em roteamento com restrições (*Constraint-based routing*) são utilizadas. Nesse tipo de roteamento, os caminhos por onde pacotes de informação irão trafegar são determinados considerando-se vários tipos de restrições de tal forma a atender os requisitos de qualidade de serviço exigidos pelas aplicações.

Por esses motivos, a Internet que hoje utilizamos tem uma sucessora no horizonte, a Internet 2, que distingue-se da rede atual em dois aspectos fundamentais: rapidez e qualidade. Essas diferenças são devidas, sobretudo, à adoção do IPv6 (Internet Protocol version 6), ao invés do IPv4. O IPv6 usa um sistema de transporte de informação mais avançado, designado multicasting (diferente do atual unicasting). Na prática, o multicasting é uma tecnologia que reduz o tráfego na rede, diminuindo drasticamente o consumo de largura de banda: um mesmo pacote de informação é enviado para múltiplos destinatários simultaneamente, uma vez que uma cópia da informação fica no roteador, que depois a envia para os vários destinos. No unicasting, um mesmo pacote é enviado várias vezes, de forma que as ligações serão tantas quanto o número de destinatários. Desse modo, o IPv6 permite que aplicações "pesadas" (que contenham dados, som e imagem, como transmissões diretas de concertos ao vivo, de conferências e de intervenções cirúrgicas, etc) convivam umas com as outras sem problemas e que a informação seja disponibilizada, de fato, em tempo real.

Além de ser uma rede com uma maior largura de banda, a Internet 2 é também mais confiável. O IPv6 atende os requisitos exigidos pelos padrões QoS (Q*uality of Service*). Isso significa que a Internet 2 permite que diferentes tipos de tráfego e de aplicações recebam tratamento distinto na rede. Os padrões QoS incluem, nomeadamente, as definições de largura de banda constante (essencial na transmissão de vídeo) e reserva de largura de banda (espaço dedicado a certo tipo de aplicações), baixa latência (tempo de espera mínimo para os pacotes de informação e redução dos atrasos) e perda mínima de pacotes de informação em trânsito.

Dentro deste novo contexto, existe a necessidade de estabelecer o dimensionamento da rede, alocando os recursos de acordo com a demanda apresentada.

Assim, o objetivo do dimensionamento de uma rede IP é determinar a alocação dos recursos que satisfaça os requisitos de qualidade de serviço e engenharia de tráfego.

A estimativa do tráfego que irá passar pela rede é uma das entradas necessárias quando se deseja dimensioná-la. Dois modelos de especificações de tráfego são de grande importância dentro deste contexto: o modelo *pipe* (tubo), que descreve o fluxo de tráfego entre um nó de entrada e um nó de saída e o modelo *hose* (funil) que descreve o fluxo entre um nó de entrada e um ou mais nós de saída [Goderis et al., 2000]. Como Serviços Diferenciados lida com agregado de dados, o modelo *hose* se mostra adequado para especificar tráfegos nessa arquitetura.

O dimensionamento da rede resolve o problema de encontrar o conjunto de caminhos que satisfaçam os requisitos de qualidade de serviço e de engenharia de tráfego estabelecidos, utilizando para isto os dados relativos a uma topologia, como as características dos enlaces pertences a ela (recursos disponíveis) e um conjunto de especificações de agregados de tráfegos entre o nó de origem e os nós de destino da rede (estimativa de tráfego).

Sob a ótica da otimização, o problema de encontrar rotas que satisfaçam a esses requisitos pertence à categoria NP-dificil, o que significa que possui ordem de complexidade combinatorial. Em outras palavras, o esforço computacional para a sua resolução cresce exponencialmente com o tamanho do problema (dado pelo número de nós).

Em termos práticos, isto significa que, na resolução de problemas reais de otimização pertencentes à classe NP-dificil, raramente são encontrados os resultados ótimos. Conseqüentemente, os métodos de solução aplicados a instâncias reais são, em geral, heurísticas, isto é, tais métodos não asseguram a obtenção da solução ótima do ponto de vista matemático. Dentre esses métodos destacam-se as metaheurísticas, as quais, ao contrário das heurísticas convencionais, são providas de mecanismos para escapar de ótimos locais, além de serem de fácil implementação e permitir incluir, com facilidade, múltiplas restrições no roteamento.

Este trabalho apresenta a formulação e implementação de algoritmos baseados nas técnicas GRASP (*Greedy Randomized Search Procedure*) [Feo & Resende, 1995] e Busca Tabu [Glover, 1986] para resolver o problema de tráfego na rede Internet gerando uma melhora de desempenho da rede.

2 REFERENCIAL TEÓRICO

Dentre as inovações tecnológicas mais recentes, a *Internetwork* ou simplesmente *Internet*, como usualmente é denominada, representa um dos avanços mais significativos na área de informação e comunicação da sociedade globalizada. Rompendo com os limites impostos pelo espaço e tempo, a Internet oferece aos seus usuários múltiplas possibilidades de interação com a comunidade mundial em tempo real, isto é, acontecimentos podem ser transmitidos simultaneamente à sua ocorrência.

Ainda que se afirme que a Internet representa um dos principais espaços de democratização de acesso à informação, isso ainda está longe de ser verdadeiro, em função dos custos de acesso. Entretanto, não há como negar a sua popularização e, essa crescente expansão do número de usuários tem gerado problemas que precisam ser solucionados; por exemplo, atraso, confiabilidade, etc. Tais problemas referem-se à transmissão de dados entre emissor e receptor, com as necessárias e justas garantias.

Paralelamente ao aumento da demanda, as inovações tecnológicas têm exigido da rede Internet um desempenho que ultrapassa a capacidade de funcionamento do modelo vigente. Hoje, trafegam pela Internet não só pacotes de dados como também de vídeo e voz, haja visto que as vídeo conferências já estão se tornando usuais.

Portanto, atender às tais demandas, propiciando a oferta de produtos e serviços de forma eficaz e cada vez mais eficiente, implica em propor soluções para os atuais problemas que se anunciam em relação ao tráfego de informações na principal e mais moderna rede de comunicações que é a Internet.

O modelo atual da Internet baseia-se num único nível de serviço, de modo que os pacotes de dados que trafegam pela rede estão igualmente sujeitos a atrasos, perdas, alteração da ordem ou mesmo descarte, quando as filas estão cheias, independente de seu nível de importância.

Dada a crescente expansão comercial da Internet e as demandas decorrentes de novas aplicações, que exigem mais eficiência da rede, inúmeros estudos estão sendo desenvolvidos com vistas a melhorar a qualidade dos serviços oferecidos.

Nesse contexto, serão discutidos a seguir:

- Qualidade de Serviços na Internet (QoS) [Ferguson e Huston, 1998], que se refere ao desempenho da rede proporcionando as garantias de serviço aos usuários da rede;
- Arquitetura de Serviços Integrados (IntServ) [Braden et al., 1994], que utiliza reservas de recursos para fluxos individuais;

- Arquitetura de Serviços Diferenciados (DiffServ) [Blake et al., 1998], que utiliza o conceito de agregação de fluxos;
- Multiprotocol Label Switching (MPLS) [Rosen & Callon, 2001], que consiste na utilização de rótulos de tamanho fixo para encaminhamento de pacotes de dados;
- Engenharia de Tráfego (TE), que define como os fluxos atravessam a rede.
- Acordo e Especificação em Nível de Serviço [Trimintzios et al., 2001], que determinam os termos e cláusulas de contratos entre um provedor e seus consumidores;
- Projeto TEQUILA (Traffic Engineering for Quality of Service in the Internet at Large Scale) [Goderis et al., 2000], que tem por objetivo estudar, especificar, implementar e validar um conjunto de definições de serviço e ferramentas de Engenharia de Tráfego para obter garantias quantitativas de QoS fim-a-fim;
- GRASP (Greedy Randomized Adaptive Search Procedure) [Feo & Resende, 1995] e Busca Tabu [Glover, 1986], que são metaheurísticas, as quais, ao contrário das heurísticas convencionais, são providas de mecanismos para escapar de ótimos locais, além de serem de fácil implementação e permitir incluir, com facilidade, múltiplas restrições no roteamento.

2.1 Qualidade de Serviços (QoS)

Qualidade de Serviços aplicado à rede de computadores refere-se tanto ao desempenho de uma rede em relação às necessidades de aplicações quanto ao conjunto de tecnologias que possibilitam às redes oferecer garantia de desempenho [Teitelman, 1998 apud Kamienski]. As garantias de desempenho fim-a-fim são avaliadas entre os nós de ingresso e de egresso de dados na rede, a partir de 4 parâmetros [Ferguson e Huston, 1998]:

- Latência: é o tempo gasto pelos pacotes de dados para trafegar do emissor até o receptor. Aplicações como vídeo e áudio apresentam tempo de latência alto. A latência é também designada como “atraso” nas literaturas afins.
- Variação do retardo (jitter): é a variação no atraso fim-a-fim. Variações acentuadas no retardo podem comprometer a qualidade do serviço oferecido a algumas aplicações.

- Vazão: é a taxa efetiva de transferência de dados entre dois nós da rede. Esse parâmetro é influenciado pela largura da banda (capacidade física da banda) e pela quantidade de fluxos compartilhados.
- Confiabilidade: capacidade da rede em entregar corretamente os dados. No percurso entre a origem e o destino, os pacotes podem seguir caminhos distintos, estando sujeitos a atrasos, alteração de sua ordem, ou mesmo serem descartados, quando as filas estão cheias.

A partir desses parâmetros é possível afirmar que um serviço de qualidade exige necessariamente baixa latência, baixa variação do retardo, alta vazão e alta confiabilidade. As possibilidades de viabilização das melhorias da qualidade dos serviços da Internet apontam para uma diferenciação criando categorias de tráfego.

Segundo Kamienski [1999]:

> *"o serviço oferecido pela Internet é justo para com todos os usuários, mas o que se quer, na realidade, é introduzir uma boa dose de injustiça, a fim de beneficiar usuários ou aplicações que desejam ou podem pagar por serviços de melhor qualidade."*

Para levar a qualidade de serviços (QoS) oferecidos pela Internet, a *Internet Engineering Task Force* (IETF) propôs alguns modelos e mecanismos possíveis de serem implementados. Entre eles destacam-se a *Arquitetura de Serviços Integrados (IntServ)* e a *Arquitetura de Serviços Diferenciados (DiffServ)*.

2.2 Arquitetura de Serviços Integrados (IntServ)

A Arquitetura de Serviços Integrados [Braden et al., 1994] parte do princípio de que a atual arquitetura da Internet não precisa ser modificada, mas pode ser estendida para fornecer vários novos serviços. Nesse modelo há a necessidade de reservas de recursos e controle de admissão, de modo que em grandes redes isso se torna inviável em função da necessidade de manter recursos reservados em todos os roteadores.

2.3 Arquitetura de Serviços Diferenciados (DiffServ)

A Arquitetura de Serviços Diferenciados [Blake et al., 1998] parte da idéia central de manter informação de estado (state information) e complexidade somente nas extremidades da rede, enquanto os roteadores de trânsito são responsáveis por aplicar tratamento apropriado no repasse dos pacotes de acordo com o seu campo de serviços diferenciados do cabeçalho IP (Internet Protocol). As redes que implementam serviços diferenciados são chamadas Domínios de Serviços Diferenciados (DS). Um domínio é composto por um conjunto de nós que compartilham uma mesma política de serviços.

Pacotes distintos podem ter tratamentos distintos nos roteadores para manter sua conformidade com os requisitos QoS de sua classe. Esse tratamento específico de encaminhamento é chamado de *PHB* (*Per-Hop Behavior*), ou seja, representa um conjunto de ações feitas pelos roteadores, tais como política de escalonamento, buffering, que define o comportamento de um agregado na rede que implemente o DiffServ.

Três tipos de comportamento (PHB) foram propostos:

- O PHB *Encaminhamento Expresso* (*Expedited Forwarding - EF*) [Jacobson et al., 1999]: define garantias quantitativas de QoS para suportar aplicações muito sensíveis a variações de natureza temporal da rede. Exemplos de serviços que utilizam o PHB EF incluem emulação de circuitos e serviços de voz e vídeo.
- O PHB *Encaminhamento Assegurado* (*Assured Forwarding - AF*) [Heinamem et al., 1999]: define garantias qualitativas de serviços, estabelecendo preferências de pacotes fluindo na rede.
- O PHB *Melhor Esforço* (*Best Effort - BE*): reflete um comportamento sem nenhuma garantia, correspondendo ao melhor esforço para transmissão, tal qual a Internet funciona hoje.

De acordo com os tipos de comportamento, os pacotes podem ser agrupados em conjuntos, denominados *Agregado de Tráfego* (*Traffic Aggreate - TA*) e *Agregado de Ordem* (*Ordered Aggregate - OA*).

O Agregado de Tráfego corresponde ao agrupamento de pacotes que recebem um mesmo código, denominado DSCP (Differenciated Service Code Point). Por exemplo, “agregado de tráfego EF”, corresponde ao conjunto de pacotes PHB EF.

O Agregado de Ordem corresponde ao conjunto de agregados de tráfego que compartilham uma restrição de ordenação, ou seja, uma restrição imposta pela sua prioridade em relação aos demais.

2.4 Multiprotocol Label Switching (MPLS)

O MPLS é um protocolo baseado em rótulos (Label Switching) diferente do protocolo IP utilizado na Internet atual.

No roteamento IP, cada pacote é analisado a partir do cabeçalho IP do pacote e, de acordo com a busca na tabela de roteamento, ocorre o encaminhamento. Essa busca pode ter um tempo de processamento elevado, dependendo do tamanho da tabela do roteador.

O MPLS consiste em encaminhamento de pacotes IP baseado em rótulos (labels) ao invés de endereços. Para tal, foi idealizada a Arquitetura *MPLA* (*Multiprotocol Label Architecture*) [Rosen and Callon, 2001], que permite o estabelecimento de uma rota explícita entre um nó de ingresso e um nó de egresso, denominado *ER-LSP* (*Explicitly Routed – Label Switched Path*) ou simplesmente LSP.

Numa rota LSP, um pacote IP é analisado pelos roteadores de borda da rede criando um caminho para o mesmo e atribuindo-lhe um rótulo. Ou seja, os LSP, dentro da arquitetura utilizada são configurados na abertura da conexão usando o protocolo MPLS. Quando um roteador recebe um pacote com cabeçalho MPLS, o rótulo presente no cabeçalho é usado como chave primária de consulta à tabela de encaminhamento MPLS presente no roteador, onde são recuperados a porta de destino e o novo rótulo do pacote, que foram definidos quando a rota foi estabelecida. Em outras palavras, os demais roteadores irão substituir sucessivamente os rótulos ate que o pacote chegue ao destino.

Segundo Rabbat e Laberteaux [2000], "o MPLS é um protocolo capaz de especificar ou 'fixar' a rota de um fluxo provendo garantias quantitativas". O MPLS fixa uma rota particular para um fluxo determinado por um processo de alocação de recursos da rede. Desse modo, o MPLS introduz o conceito de caminho virtual e implementa a conexão no nível de IP.

É importante ressaltar ainda que, segundo Goderis et al. [2001], fluxo de tráfego é definido como o fluxo de pacotes entre dois nós que possui características próprias e é tratado pela rede de um modo único e pré-estabelecido. Micro-fluxo é entendido como o

fluxo de pacotes que possuem os mesmos endereços de origem e destino, portas de origem e destino e identificação de protocolo.

O conceito de Tronco de Tráfego (TT) está relacionado com a agregação de fluxos de tráfego pertencentes à mesma classe, que são encaminhados através de uma rota (LSP) comum. Um tronco de tráfego pode ser caracterizado por um nó de ingresso e um nó de egresso, e um conjunto de atributos que determinam seu comportamento e requisitos na rede. A partir desse conceito, define-se matriz de tráfego como uma representação da demanda de tráfego entre um conjunto de nós de origem e destino [Goderis et al., 2001].

Dois modelos de especificação de tráfego foram propostos. O modelo *pipe* (tubo) descreve o fluxo de tráfego entre um nó de ingresso e um nó de egresso, sendo assim, um tronco de tráfego é associado a esse par de nós. O modelo *hose* (funil) descreve o fluxo de tráfego entre um nó de ingresso e um ou mais nós de egresso. Isso implica que os fluxos entre os nós de ingresso e egresso devem ser convergidos em uma mesma alocação de fluxo de dados, até que seja necessária a sua divergência. Dessa forma, o conceito de tronco de tráfego é estendido para ser associado a um nó de ingresso e vários nós de egresso, de modo que um fluxo de dados que entra na rede por um nó de ingresso pode ser direcionado para qualquer um dos seus nós de egresso. O uso do modelo *hose* ao invés do modelo *pipe* é de fundamental relevância no atual contexto tecnológico. A multiplexação dos fluxos na rede possibilita uma otimização do fluxo de dados, diminuindo a banda de passagem reservada na rede [Kumar et al., 2001].

2.5 Engenharia de Tráfego

A engenharia de tráfego da Internet é definida como a área da Engenharia de Rede Internet que trata das questões relativas à avaliação de desempenho e à otimização das redes operacionais IP, por meio da aplicação de tecnologia e princípios científicos para a quantificação, caracterização, modelagem e controle do tráfego da Internet [Awduche et al., 1999] .

Em função do aumento do número de usuários e das novas aplicações multimídias, a atual tecnologia da Internet já não garante ao usuário um nível de qualidade de serviço adequado. Hoje, a arquitetura da Internet exige mudanças no seu modo de funcionamento e, observa-se que, a qualidade de serviço em uma rede IP está

diretamente relacionada com a otimização de seu desempenho. Em outras palavras, melhorar a qualidade dos serviços oferecidos pela Internet significa investir na área de Engenharia de Tráfego.

Na atual versão da Internet, as operações de Engenharia de Tráfego não são definidas, de modo que torna-se impossível o estabelecimento de rotas explícitas entre os nós de uma rede. Por exemplo, para atender aos requisitos de qualidade de serviço feitos por uma aplicação, poderia ser necessário redirecionar o tráfego não prioritário para outras rotas e redirecionar o tráfego prioritário para as rotas que melhor atendam as restrições e garantias exigidas pela aplicação.

Rabbat e Laberteaux [2000] afirmam que "usando a funcionalidade de DiffServ e acrescentando a esse a funcionalidade roteamento-fixo proporcionado por MPLS, é possível garantir QoS quantitativas em redes como a Internet".

2.6 Acordo e Especificação em Nível de Serviço

Os serviços com QoS são oferecidos através de acordos (Service Level Agreements - SLA), definidos em termos e cláusulas de contratos entre um provedor e seus consumidores. Estes termos e cláusulas acordados são convertidos em um conjunto padrão de especificações em nível de serviço (Service Level Specification - SLS). O conteúdo de uma especificação inclui os parâmetros essenciais relacionados a QoS, incluindo identificação de escopo e de fluxo, parâmetros de conformação de tráfego e garantias de serviço [Trimintzios et al., 2001].

Para estender a arquitetura IP, em uso na Internet, é preciso adicionar alguns mecanismos, descritos a seguir.

- Um mecanismo de especificação de tráfego de entrada e das garantias de despacho asseguradas pela rede. Para tal, acordos de nível de serviço (SLAs) devem ser firmados e especificações de nível de serviço (SLSs) devem estabelecer patamares de garantias que devem ser honrados pela rede.
- Mecanismos de diferenciação do despacho de pacotes de acordo com a qualidade de serviço desejada e previamente acordada nos SLAs firmados. Para tal, a arquitetura de Serviços Diferenciados deve implementar tal funcionalidade em IP.

- Um mecanismo de roteamento que possibilite a escolha das rotas para satisfazer os requisitos de qualidade de serviço e engenharia de tráfego. Para tal, faz-se necessário determinar rotas para determinados tipos de tráfego e MPLS é uma ferramenta que implementa rotas explícitas em IP.

Para atender o funcionamento dos mecanismos acima de forma harmoniosa, foram idealizadas arquiteturas como as propostas pelos projetos TEQUILA [Goderis et al., 2000] e Qbone [Teitelbaum, 2000].

2.7 Projeto TEQUILA

O Projeto TEQUILA (Traffic Engineering for Quality of Service in the Internet at Large Scale) tem por objetivo estudar, especificar, implementar e validar um conjunto de definições de serviço e ferramentas de Engenharia de Tráfego para obter garantias quantitativas de QoS fim-a-fim através de planejamento cuidadoso, dimensionamento e controle dinâmico de técnicas de gerenciamento de tráfego simples e escalonável (segundo hierarquias) na Internet (i.e. DiffServ) [Goderis et al., 2000].

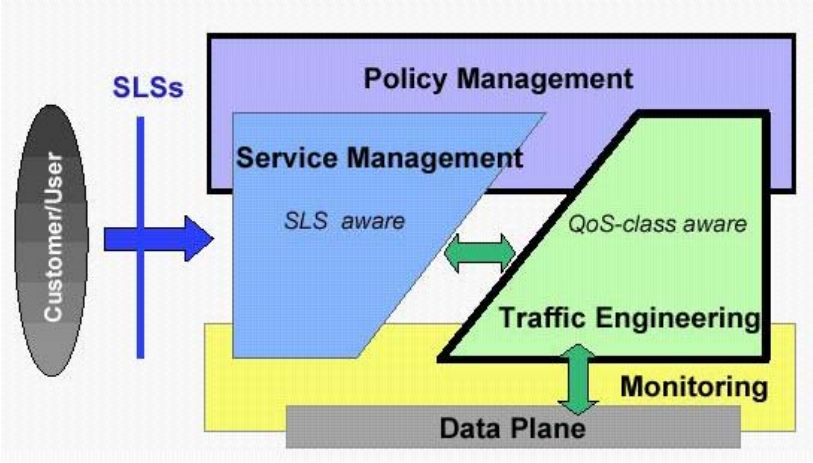


Figura 1: Arquitetura Funcional TEQUILA

A arquitetura funcional do projeto Tequila (Figura 1) é dividida em duas partes, o cliente (*Customer*) e o provedor de serviços (*Provider*). A ligação entre as duas partes é feita através do acordo de garantias de QoS tratado. No provedor estão definidos quatro blocos funcionais: Gerenciador de Políticas (*Policy Management*), responsável por tratar das políticas de utilização da rede; Gerenciador de Serviços (*Service Management*), onde são controlados os serviços acordados (SLS) e realizada a previsão

do tráfego a ser alocado na rede; Engenharia de Tráfego (*Traffic Engineering*), que tem como finalidade definir as configurações da rede para atender aos SLSs e as políticas de rede, gerenciando os dispositivos da rede e realizar o roteamento na rede; Monitoramento (*Monitoring*), analisa a transmissão dos dados na rede, verificando se estão sendo atendidos satisfatoriamente os SLSs e fornece uma análise sobre os dados trafegados na rede.

2.7.1 Bloco Dimensionamento de Rede

O bloco dimensionamento de rede encontra-se inserido no bloco de Engenharia de Tráfego da Arquitetura TEQUILA, e tem a função de prover uma efetiva solução de engenharia de tráfego, determinando a alocação adequada de recursos, considerando as tendências de tráfego, os requisitos de qualidade de serviço e diretivas de política. As entradas para o bloco são a matriz de tráfego fornecida pelo bloco previsão de tráfego (TF – Traffic Forecast) e a topologia de rede; a partir desses dados e das diretivas de política correntes, é possível calcular o conjunto de rotas explícitas (ER-LSP) e determinar os parâmetros para a configuração da rede [Cavalcanti, 2000]. A figura 2 mostra o esquema descrito.

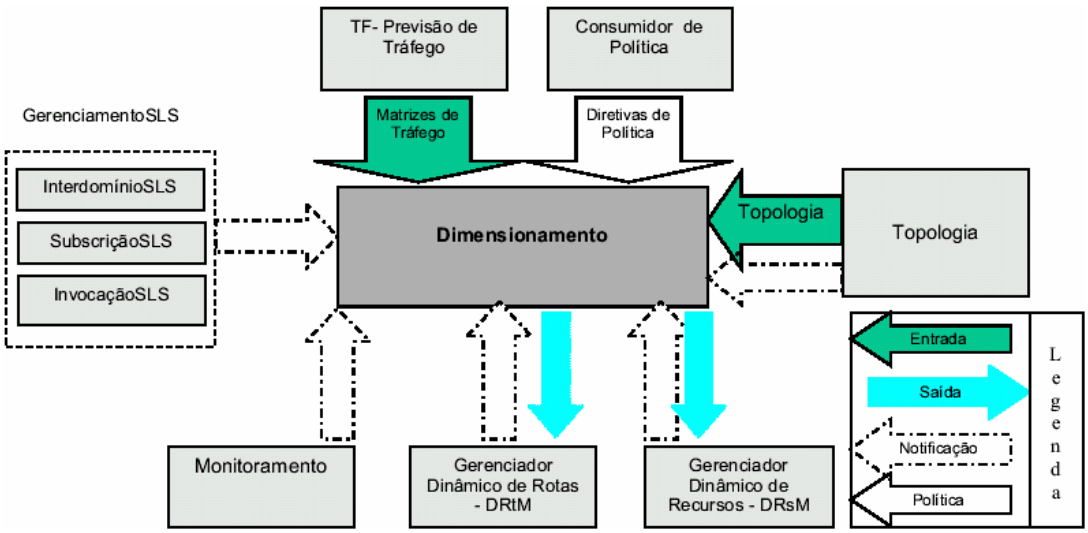


Figura 2: Interface entre o bloco dimensionamento de rede e outros blocos funcionais

O bloco de dimensionamento funciona tanto para uma engenharia de tráfego a longo prazo (meses), chamada Planejamento de Rede quanto para a curto prazo chamada Dimensionamento de Rede (horas-semanas).

Esse planejamento é realizado sobre uma previsão de demanda e sobre uma configuração de rede, sujeita à falhas. Portanto, é necessário que o módulo de dimensionamento de rede seja capaz de computar rapidamente novas LSPs quando por algum motivo as existentes não forem mais viáveis.

O módulo de Previsão de Tráfego (*Traffic Forecast* - TF) estima uma demanda de tráfego orientada aos SLS (SLS *aware*) e em relação a informações obtidas do bloco de monitoramento de rede (SLS *unaware*). Esta demanda é representada por uma matriz de troncos de tráfego, agrupados por agregados de ordem.

O módulo Gerenciador Dinâmico de Rotas (*Dynamic Route Managemen*t – DRtM), baseado nas restrições e regras providas pelo bloco de dimensionamento de rede, modifica os parâmetros nos roteadores, além de realizar ajustes dinâmicos, como adicionar, juntar, separar e re-configurar os caminhos explícitos, adequando pequenas variações de tráfego à rede.

O módulo Gerenciador Dinâmico de Recursos (*Dynamic Resource Management* – DRsM) é responsável pelo controle dos recursos da rede. Sua interação com o módulo de dimensionamento de rede se faz informando sobre os recursos disponíveis para alocação dos troncos de tráfego ou notificando a necessidade de recalcular as rotas por alguma alteração na disponibilidade da rede como, por exemplo, a falha de um enlace na rede.

2.8 GRASP (Greedy Randomized Adaptive Search Procedure)

GRASP (Procedimento de busca adaptativa gulosa e randomizada) é um método iterativo, proposto por Feo & Resende [1995], que consiste de duas fases: uma fase de construção, na qual uma solução é gerada, elemento a elemento, e de uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado. O pseudocódigo ilustrado pela Figura 3 descreve o procedimento GRASP.

```
procedimento GRASP (f(.), g(.), N(.), GRASPmax, s)
1.  f* ← ∞;
2.  para (Iter = 1,2,...,Graspmax) faça
3.      Construção (g(.),α,s);
4.      BuscaLocal(f(.),N(.),s);
5.      se (f(s) < f*) então
6.          s* ← s;
7.          f* ← f(s);
8.   fim-se;
9.  fim-para;
10. s ← s*;
11. Retorne s;
fim GRASP;
```

Figura 3: Algoritmo GRASP

Na fase de construção, uma solução é iterativamente construída, elemento por elemento. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordenação pré-determinado. Esse processo de seleção é baseado em uma função adaptativa gulosa $g : C \rightarrow \Re$, que estima o benefício da seleção de cada um dos elementos.

A heurística é adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças oriundas da seleção do elemento anterior. A componente probabilística do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos. Este subconjunto recebe o nome de lista de candidatos restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração GRASP.

Seja $\alpha \in [0,1]$ um dado parâmetro. O pseudocódigo representado pela Figura 4 descreve a fase de construção GRASP. O parâmetro α controla o nível de gulosidade e aleatoriedade do procedimento Construção. Um valor $\alpha = 0$ faz gerar soluções puramente gulosas, enquanto $\alpha = 1$ produz soluções totalmente aleatórias.

procedimento *Construção* (g(.), α, s);

1. $s \leftarrow \varnothing$;
2. Inicialize o conjunto C de candidatos;
3. enquanto ($C \neq \varnothing$) faça
4. $t_{min} = \min\{g(t) \mid t \in C\}$;
5. $t_{max} = \max\{g(t) \mid t \in C\}$;
6. $LCR = \{ t \in C \mid g(t) \leq t_{min} + \alpha(t_{max} - t_{min}) \}$;
7. Selecione, aleatoriamente, um elemento $t \in LRC$;
8. $s \leftarrow s \cap \{t\}$;
9. Atualize o conjunto C de candidatos;
10. fim-enquanto;
11. Retorne s;

fim C*onstrução;*

Figura 4: Fase de Construção do Algoritmo GRASP

O procedimento GRASP procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos, com aqueles dos procedimentos aleatórios de construção de soluções [Souza, 2002].

2.9 Métodos de Busca Local

As técnicas de busca local em problemas de otimização constituem uma família de técnicas baseadas na noção de vizinhança. Mais especificamente, seja S o espaço de pesquisa de um problema de otimização e f a função objetivo a minimizar. A função N , a qual depende da estrutura do problema tratado, associa a cada solução viável $s \in S$, sua vizinhança $N(S) \subseteq S$. Cada solução $s' \in N(s)$ é chamada de vizinho de s. Denomina-se movimento a modificação m que transforma uma solução s em outra, s' , que esteja em sua vizinhança. Representa-se essa operação por $s' \leftarrow s \oplus m$. Em linhas gerais, uma técnica de busca local, começando de uma solução inicial s_0 (a qual pode ser obtida por alguma outra técnica ou gerada de forma aleatória), navega pelo espaço de pesquisa, passando, iterativamente, de uma solução para outra que seja sua vizinha [Souza, 2002].

2.9.1 Método de Descida

É um método de busca local que se caracteriza por analisar todos os possíveis vizinhos de uma solução s em sua vizinhança N(s), escolhendo, a cada passo, aquele que tem o menor valor para a função de avaliação. Nesse método, o vizinho candidato somente é aceito se ele melhorar estritamente o valor da melhor solução até então

obtida. Dessa forma, o método pára tão logo um mínimo local seja encontrado [Souza, 2002].

2.9.2 Busca Tabu

Os princípios básicos da Busca Tabu (BT), técnica originada nos trabalhos de Fred Glover [1986] e Pierre Hansen [1986], são descritos a seguir. A Busca Tabu é um procedimento adaptativo que utiliza uma estrutura de memória para guiar um método de descida a continuar a exploração do espaço de soluções mesmo na ausência de movimentos de melhora, evitando que haja a formação de ciclos, isto é, o retorno a um ótimo local previamente visitado.

Mais especificamente, começando com uma solução inicial *s0* , um algoritmo BT explora, a cada iteração, um subconjunto V da vizinhança N(*s*) da solução corrente *s* . O membro *s'* de V com menor valor nessa região segundo a função f(.) torna-se a nova solução corrente mesmo que *s0* seja pior que *s* , isto é, que f(*s'*) > f(*s*).

O critério de escolha do melhor vizinho é utilizado para escapar de um mínimo local. Esta estratégia, entretanto, pode fazer com que o algoritmo cicle, isto é, que retorne a uma solução já gerada anteriormente. De forma a evitar que isto ocorra, existe uma lista tabu *T* , a qual é uma lista de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos |*T*| movimentos realizados (onde |*T*| é um parâmetro do método) e funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo sai.

Assim, na exploração do subconjunto *V* da vizinhança N(*s*) da solução corrente *s*, ficam excluídos da busca os vizinhos *s'* que são obtidos de *s* por movimentos *m* que constam na lista tabu. A lista tabu se, por um lado, reduz o risco de ciclagem (uma vez que ela garante o não retorno, por |*T*| iterações, a uma solução já visitada anteriormente); por outro, também pode proibir movimentos para soluções que ainda não foram visitadas. Assim, existe também uma função de aspiração (*A*), que é um mecanismo que retira, sob certas circunstâncias, o status tabu de um movimento.

Duas regras são normalmente utilizadas de forma a interromper o procedimento. Pela primeira, pára-se quando é atingido um certo número máximo de iterações sem melhora no valor da melhor solução. Pela segunda, quando o valor da melhor solução chega a um limite inferior conhecido (ou próximo dele). Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução ótima é encontrada ou quando uma solução é julgada suficientemente boa.

Os parâmetros principais de controle do método de Busca Tabu são a cardinalidade $|T|$ da lista tabu, a função de aspiração A , a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e *BTmax*, o número máximo de iterações sem melhora no valor da melhor solução.

Apresenta-se, pela Figura 5, o pseudocódigo de um algoritmo de Busca Tabu básico.

```
procedimento BT (f(.), N(.), A(.), |V|, |T|, BTmax, s);
12. s* ← s;              {melhor solução obtida até então}
13. Iter ← 0;            {contador do número de iterações}
14. MelhorIter ← 0;      {iteração mais recente que forneceu s*}
15. T ← ∅;               {Lista Tabu}
16. Inicialize a função de aspiração A;
17. enquanto (Iter – MelhorIter < BTmax) faça
18.     Iter ← Iter + 1;
19.     Seja s' ← s ⊕ m o melhor elemento de V ⊂ N(s) tal que o
        movimento m não seja tabu (m ∉ T) ou s' atenda a função de aspiração
        (f(s') < A(f(s)));
20.     T ← T – {movimento mais antigo} + {movimento que gerou s'};
21.     Atualize a função de aspiração A;
22.     s ← s';
23.     se (f(s) < f(s*)) então
24.         s* ← s;
25.         MelhorIter ← Iter;
26.     fim-se;
27. fim-enquanto;
28. s ← s*;
29. Retorne s;
fim BT;
```

Figura 5: Pseudo-código do Algoritmo Busca Tabu

3 DESCRIÇÃO DO PROBLEMA

Uma rede é modelada como um grafo direcionado $G(V,E)$, onde V é um conjunto de nós, representando os roteadores, e E é o conjunto das arestas representando os enlaces da rede. Um elemento $a_{ij} \in E$ é especificado pelo par $l=(v_{i,ingress}, v_{j,egress})$, onde $v_{i,ingress}$ e $v_{j,egress}$ são os nós onde o fluxo de tráfego entra e sai na rede respectivamente, também denominados nós de ingresso e egresso.

Sendo uma classe de qualidade de serviço h implementada pelo PHB do tipo h, temos que cada enlace l possui os seguintes recursos associados com o PHB do tipo h:

1. A capacidade do enlace C^h_l;

2. O atraso do enlace d^h_l ;

O atraso total do enlace pode ser decomposto em quatro componentes distintos: processamento, enfileiramento, transmissão e propagação [Bertsekas & Gallager, 1992]. Dependendo das características associadas a um PHB, o atraso assumido pode ser determinístico ou probabilístico, máximo ou médio. Por exemplo, considerando os serviços implementados pelos comportamentos dos tipos EF e AF, num serviço implementado pelo comportamento do tipo EF, o atraso máximo deve ser determinístico e limitado a um valor máximo. Diferentemente, num serviço implementado pelo comportamento do tipo AF, o atraso pode assumir um valor esperado, isto é, com a maior probabilidade de acontecer. Geralmente, a capacidade C^h_l é expressa em Mbps e o atraso d^h_l em milisegundos.

Um tronco de tráfego $TT^h_i \in OA^h$, com nó de ingresso $v_{TT,ingress}$ e o conjunto de nós de egresso $V_{TT,egress}$, sendo $V_{TT,egress} \subseteq (V \setminus v_{TT,ingress})$, é associado a requisitos de banda de passagem de ingresso mínima e máxima, denotadas respectivamente por $b_{TT,ingress}$, $\overline{b_{TT,ingress}}$ e com um atraso fim-a-fim máximo $d_{TT,i}$. O atraso máximo pode ser determinístico ou probabilístico, dependendo do tipo do PHB h. Para cada nó de egresso $v_{TT,y} \in V_{TT,egress}$, a banda de passagem requisitada de egresso é definida como $b_{TT,y}$.

Deve-se determinar no grafo G os caminhos para cada par de nós ingresso-egresso ($v_{TT,ingress}$, $v_{TT,engress}$) dos troncos de tráfego TT^h_i, atendendo aos requisitos, objetivando minimizar a utilização da banda de passagem em todos os enlaces.

Um caminho é um conjunto ER (rota explícita), composto por enlaces l=($v_{i,ingress}$, $v_{j,egress}$), portanto ER $\subseteq$ A. Um conjunto de caminhos com o mesmo nó inicial de ingresso ($v_{TT,ingress}$), para um PHB h de qualidade de serviço, define uma árvore T^h_{TT}, cuja raiz é o nó de ingresso.

A abordagem do modelo *hose* é utilizada para determinar de modo eficiente os caminhos para os troncos de tráfego, proporcionando uma melhor utilização dos enlaces do grafo.

Portanto, defini-se o problema como: determinar uma árvore T^h_{TT} para cada tronco de tráfego $TT^h_i \in OA^h$, de modo que estas árvores atendam aos requisitos de cada TT^h_i e requisitos de largura de banda disponível em cada enlace C^h_l objetivando minimizar a utilização dos enlaces em todo o grafo.

4 MATERIAL E MÉTODOS

4.1 Implementação

A implementação do problema foi desenvolvida através do uso do software Network Simulator (NS). O pacote do NS encapsula vários módulos, com destaque para a base para criação de grafos conhecida como Standford GraphBase (SGB) e para o gerador de topologias de rede, GT Internetwork Topology Models (GT-ITM).

4.1.1 Standford GraphBase (SGB)

Trata-se de uma coleção de programas e dados de alta portabilidade, disponível para o uso em estudos de algoritmos combinatoriais e estruturas de dados.

O núcleo do SGB é composto por quatro módulos de programas.

- GB_FLIP é um gerador de números aleatórios.

- GB_GRAPH define estruturas de dados padrão para grafos e inclui rotinas para distribuição de armazenagem.

- GB_IO é responsável por ler e gravar arquivos de dados e garantir que não estejam corrompidos.

- GB_SORT é uma rotina de ordenação para chaves de 32 bits em listas ligadas de nós.

4.1.1.1 O módulo GB_GRAPH

a) Conceito

Este é o módulo de estruturas de dados usado por todas as rotinas do SGB para alocar memória. Os tipos de dados básicos para representação de grafos também são definidos neste módulo. O código é considerado independente de sistema, pois deve produzir resultados equivalentes em todos os sistemas assumindo que as funções padrão *calloc* e *free* da linguagem C estejam disponíveis.

O código C para GB_GRAPH não tem uma rotina principal (main); é composto apenas por uma porção de subrotinas que pode ser incorporada em programas em um nível mais alto via a rotina de carga do sistema.

Os programas do SGB empregam um simples e flexível conjunto de estruturas de dados para representar e manipular grafos na memória do computador. Vétices

aparecem num array seqüencial de registros *Vertex*, e os arcos provenientes de cada vértice aparecem numa lista ligada de registros *Arc*. Existe também um registro *Graph* para prover informação sobre o grafo como um todo.

O esquema estrutural para os registros *Vertex*, *Arc* e *Graph* incluem um número de campos utilitários que podem ser usados para qualquer propósito pelos algoritmos de manipulação dos grafos.

b) A Estrutura Utilitária

Cada campo utilitário é um tipo união que tanto pode ser um ponteiro de vários tipos ou um inteiro longo. Os sufixos .V, .A, .G e .S no nome e uma variável utilitária significa que a variável é um ponteiro para um vértice, um arco, um grafo ou string, respectivamente; o sufixo .I designa um inteiro longo.

```
typedef union {
    struct vertex_struct *V; /*ponteiro para um vértice*/
    struct arc_struct *A; /*ponteiro para um arco*/
    struct graph_struct *G; /*ponteiro para um grafo*/
    char *S; /*ponteiro para uma string*/
    long I; /*inteiro*/
}util;
```

c) A Estrutura Vértice

Cada vértice (Vertex) tem dois campos padrão e seis campos utilitários. Os dois campos padrão são: *arcs* (um ponteiro para um arco) e *name* (um ponteiro para uma string de caracteres).

Se v refere-se a um vértice e $v \rightarrow arcs$ é nulo (λ), não existem arcos provenientes de *v*. Caso contrário, $v \rightarrow arcs$ aponta para um registro *Arc* representado por um arco de *v*, e este registro possui um campo *next* (próximo) que aponta do mesmo modo para outro arco, para a representação de todos os outros arcos de *v*.

Os campos utilitários são chamados de *u*, *v*, *w*, *x*, *y*, *z*. Macros podem ser usadas para dar a eles uma identificação em aplicações particulares. Eles são tipicamente usados para registrar informações tais como grau de entrada e grau de saída, ou quando um vértice está "marcado". Campos utilitários podem também ligar um vértice a outros vértices ou arcos em uma ou mais listas.

```
typedef struct vertex_struct {
    struct arc_struct *arcs; /*lista ligada de arcos saindo deste vértice*/
    char *name; /*string de identificação simbólica do vértice*/
    util u, v, w, x, y, z; /*campos multi-finalidade*/
}Vertex;
```

d) A Estrutura Arco

Cada registro *Arc* tem três campos padrão e dois campos utilitários. Os campos *tip* (um ponteiro para um vértice), *next* (um ponteiro para um arco) e *len* (um inteiro longo) são padrão. Se *a* aponta para um arco na lista de arcos de um vértice *v*, ele representa um arco de tamanho $a \rightarrow len$ de *v* para $a \rightarrow tip$, e o próximo arco de *v* na lista é representado por $a \rightarrow next$. Os campos utilitários são designados por *a* e *b*.

```
typedef struct arc_struct {
    struct vertex_struct *tip; /*vértice para o qual o arco aposta*/
    struct arc_struct *next; /*próximo arco do mesmo vértice*/
    long len; /*comprimento do arco*/
    útil a, b; /*campos multi-finalidade*/
}Arc;
```

e) A Estrutura Grafo

O tipo *Graph* é uma estrutura de dados que pode ser passada para um algoritmo que opera grafos a fim de encontrar árvores de cobertura, caminhos mínimos entre outros. Um registro do tipo *Graph* tem sete campos padrão e seis campos utilitários. Os campos padrão são denotados por: *vertices* (um ponteiro para um array de registros *Vertex*), *n* (número total de vértices), *m* (número total de arcos), *id* (identificação simbólica do grafo), *util_types* (uma representação simbólica dos tipos de dados nos campos utilitários), *data* (um espaço de memória do tipo *Area* usado para armazenamento de arcos (*Arc*) e de strtings), *aux_data* (um espaço de memória do tipo *Area* usado para informação auxiliar).

Os campos utilitários são denotados por *uu*, *vv*, *ww*, *xx*, *yy* e *zz*.

```
# define ID_FIELD_SIZE 161
```

```
typedef struct graph_struct {
    Vertex *vertices; /*início do array de vértices*/
    long n; /*nº total de vértices*/
    long m; /*nº total de arcos*/
    char id[ID_FIELD_SIZE]; /*identificação na base*/
    char uitl_types[15]; /*tipo dos campos utilitários*/
    Area data; /*blocos de dados principal*/
    Area aux_data; /*blocos de dados auxiliar*/
    Util uu, vv, ww, xx, yy, zz; /*campos multi-finalidade*/
}Graph;
```

f) O campo util_types

Este campo contém uma string de comprimento 14, seguida como usual, por um caractere nulo pra indicar seu fim. Os seis primeiros caracteres especificam o uso dos campos utilitários u, v, w, x, y e z dos registros *Vertex*; os próximos dois fornecem o tipo dos campos utilitários a e b dos registros *Arc*; e os últimos seis indicam o formato dos campos utilitários dos registros *Graph*. Cada caractere pode ser: I (um inteiro longo), S (um ponteiro para uma string), V (um ponteiro para um vértice), A (um ponteiro para um arco), G (um ponteiro para um grafo), Z (um campo não utilizado, ou seja, zero). O default para util_types é "ZZZZZZZZZZZZZZ" quando nenhum dos campos utilitários está sendo usado.

A string util_types é examinada somente pelas rotinas *save_graph* e *restore_graph*, que convertem os grafos de estruturas de dados internas para arquivos externos simbólicos e vice-versa.

g) Outras rotinas

A base provê ainda muitas outras subrotinas no módulo GB_GRAPH, tais como *gb_new_graph()*, que cria um novo grafo; *switch_to_graph(Graph *)*, que permite a manipulação de mais de um grafo ao mesmo tempo, trocando o grafo corrente; *gb_recycle(Graph *)*, que apaga um grafo inteiro quando ele não é mais necessário; entre outras.

Uma tabela hash é criada para cada grafo, a fim de facilitar a busca de vértices através de seu nome. As subrotinas a seguir são responsáveis pela manipulação de tal tabela:

- *hash_in(v)*, põe o nome do vértice v na tabela hash;

- *hash_out(s)*, acha um vértice nomeado s, se presente na tabela hash;

- *hash_setup(g)*, prepara uma tabela hash para todos os vértices do grafo g;

- *hash_lookup(s,g)*, procura o nome s na tabela hash de g.

As subrotinas *hash_in* e *hash_out* aplicam-se ao grafo corrente sendo criado, enquanto *hash_setup* e *hash_lookup* aplicam-se a grafos arbitrários.

4.1.1.2 O módulo GB_DIJK

A rotina do SGB chamada *dijkstra(uu, vv, gg, hh)* encontra o caminho mais curto do vértice *uu* para o vértice *vv* no grafo *gg*, com a ajuda opcional da função heurística *hh*. Esta função implementa uma versão do algoritmo de Dijkstra, um procedimento geral para determinar caminhos mais curtos em grafos direcionados que tenham arcos de comprimento não negativo.

A idéia básica do algoritmo de Dijkstra é explorar os vértices do grafo por ordem de sua distância ao vértice inicial *uu* prosseguindo até que vv seja encontrado. Após dijkstra ter encontrado o caminho mínimo, ele retorna o comprimento do caminho. Se nenhum caminho de *uu* para *vv* existir (em particular, se *vv* é nulo), é retornado –1.

4.1.2 GT-ITM

a) Conceito

O GT_ITM (*Georgia Tech - Internetwork Topology Models*) é um pacote que contém código capaz de gerar grafos que modelam a estrutura topológica das redes. Os modelos de topologia de redes são construídos utilizando o SGB (a plataforma de estrutura de dados e rotinas para representação e manipulação de grafos).

b) Modelo de Grafos planos aleatórios

Uma variedade de grafos aleatórios não hierárquicos é encontrada na literatura de redes. Todos são variações do modelo de grafos padrão, que distribui vértices em locais aleatórios em um plano e depois considera cada par de vértices; um laço é adicionado entre um par de vértices com probabilidade α. Como ele não tem a intenção explícita de refletir a estrutura real das redes, ele é atrativo por sua simplicidade e é comumente usado no estudo de problemas de rede.

4.1.3 Network Simulator (NS)

O NS é um simulador de eventos discretos, focado para o desenvolvimento de pesquisas em redes de computadores. Ele prevê suporte a TCP e variantes do protocolo (Tahoe, Reno, New Reno, Vegas, etc.), multicast, redes sem fio (*wireless*), roteamento e satélite. Tem facilidades de *tracing*, que é a coleta e registro de dados de cada evento da simulação para análise posterior. Possui um visualizador gráfico para animações da simulação (*nam – network animator*), *timers* e escalonadores, modelos para controle de erros e algumas ferramentas matemáticas como gerador de números aleatórios e integrais para cálculos estatísticos. Inclui também uma ferramenta de plotagem, o *xgraph*, e vários tipos de geradores de tráfego.

O NS foi desenvolvido na linguagem orientada a objetos C++, de forma modular. O uso desta linguagem nos módulos confere velocidade e mais praticidade na implementação de protocolos e modificação de classes. A interface com o usuário, configuração, estabelecimento de parâmetros e manipulação de objetos e classes é feita em modo texto, através da linguagem interpretada Otcl, que também é orientada a objetos.

A utilização de simuladores para testar e validar os modelos propostos é de grande importância em diversas áreas. A adoção do NS garante o uso de uma ferramenta de ponta, podendo influir na qualidade dos resultados dos trabalhos e em sua velocidade [Portnoi & Araújo, 2002].

4.2 O Algoritmo ND

O algoritmo de dimensionamento ND (*Network Dimensioning*), proposto por Cavalcanti [2000], é o principal algoritmo de alocação de recursos em redes IP baseada na arquitetura TEQUILA e tem como estratégia a solução de um conjunto de problemas de otimização, um para cada classe de qualidade de serviço. A implementação do Módulo de Dimensionamento de Rede está dividida em duas partes, como ilustrado na Figura 6. A primeira é responsável por prover a interface com os outros módulos da arquitetura, tratamento das diretivas de política e preparar os dados para o núcleo do módulo. O núcleo do dimensionamento de rede *(ND cor*e), é chamado uma vez por cada classe de qualidade de serviço, pelo bloco dimensionamento de rede e retorna as respectivas rotas explícitas, os LSPs. O algoritmo de dimensionamento considera a prioridade das classes de qualidade de serviço, conforme definido pelas diretivas de

política. O núcleo utiliza um algoritmo de duas fases. Uma fase de alocação com uma heurística construtiva e uma fase de realocação para eliminar possíveis inviabilidades geradas na primeira fase. É considerada uma inviabilidade as sobrecargas na utilização de um enlace, ou seja, a soma das bandas de passagem de todos os troncos de tráfegos alocados em um enlace superar a capacidade deste enlace.

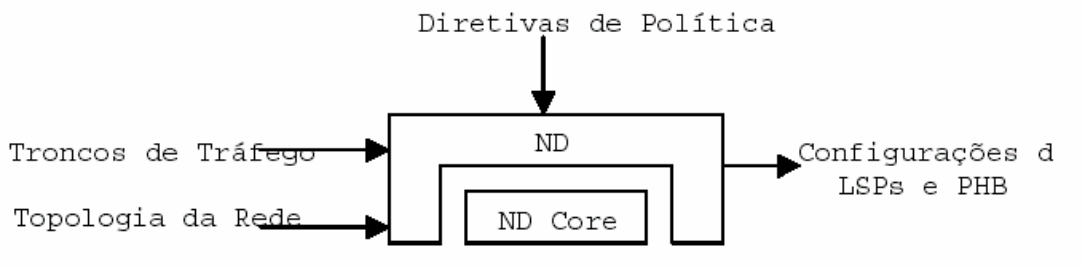


Figura 6: Implementação do módulo de dimensionamento

4.2.1 Fase de alocação do algoritmo ND

Na fase de alocação do algoritmo ND é gerada a solução inicial a partir da alocação individual de cada tronco de tráfego na rede, obedecendo a uma ordenação decrescente em relação ao somatório das bandas de passagem dos nós egressos. A alocação de cada tronco de tráfego é um procedimento heurístico, denotado por *First_Allocate*, baseado em caminhos mínimos entre os nós de ingresso e egresso do fluxo.

Os caminhos de um tronco de tráfego entre os nós de ingresso e egresso são caminhos mínimos na rede e são gerados individualmente para cada dupla. Um pré-processamento na rede é realizado para os arcos que não possuem banda de passagem disponível para atender aos requisitos não sejam utilizados. Na inexistência de um caminho que atenda aos requisitos entre os nós de ingresso e egresso são consideradas relaxações no estado presente da rede, como considerar os recursos originais da topologia, para se determinar uma nova alocação.

No caso da topologia original de rede não comportar algum tronco de tráfego por causa dos requisitos de QoS do agregado de ordem, é gerada uma exceção, sinalizando a inviabilidade na alocação da demanda de tráfego naquela topologia de rede.

No modelo *pipe* esses caminhos já são a solução. O uso do modelo *hose* nesta abordagem consiste em considerar fluxos de um tronco de tráfego em um mesmo enlace

sendo apenas um, alocando o menor valor entre a capacidade do nó de ingresso e o somatório das capacidades dos nós egressos dos fluxos analisados.

4.2.2 Fase de realocação do algoritmo ND

O objetivo desta fase é a eliminação de possíveis inviabilidades da solução construída, isto é, caso exista algum arco com alocação maior que sua capacidade é então realizado um procedimento para tentar realocar os troncos de tráfego em outros arcos. Os arcos com as maiores inviabilidades são tratados primeiro, sendo apenas uma otimização para cada arco sobrecarregado.

A otimização de um arco consiste em determinar o menor tronco de tráfego, que se realocado, elimina a inviabilidade do arco. Se a sobrecarga for maior do que a maior capacidade alocada no arco ou a realocação deste tronco de tráfego não for possível, é aplicado outro método onde se tenta realocar todos os troncos de tráfego que utilizem o arco. Uma realocação é aceita caso a nova banda de passagem utilizada pelo tronco de tráfego for menor do que a alocação prévia na rede.

Considera-se ainda, a necessidade de alocar mais recursos que a capacidade corrente do enlace, sendo sempre uma decisão de política tratar esta exceção, que é rejeitar os troncos de tráfego que causaram esta demanda ou aumentar os recursos dos enlaces [Cavalcanti, 2000].

4.3 Metodologia Proposta

4.3.1 Algoritmo GRASP aplicado ao Problema

A metaheurística GRASP apresentada na seção 2.8 foi adaptada ao problema utilizando o método `First_Allocate_GRASP` (vide seção 4.3.1.3) na fase de construção de uma solução inicial e duas variantes do método de descida, além da metaheurística Busca Tabu numa segunda abordagem, na fase de busca local (vide seção 4.3.1.4).

4.3.1.1 GRASP com Filtro

A eficiência da busca local depende da qualidade da solução construída. O procedimento de construção da solução inicial tem então um papel importante na busca local, uma vez que as soluções construídas constituem bons pontos de partida para busca local, permitindo assim não induzir a ótimos locais e podendo acelerá-la. Com essa premissa aplica-se o método de construção da solução inicial diversas vezes e

considera-se apenas a melhor dessas soluções iniciais, operando como um filtro de soluções. Visto que o custo computacional da busca local é considerado elevado no problema em questão, o refinamento é feito apenas na solução já "filtrada". A Figura 7 apresenta o pseudocódigo do procedimento GRASP com filtro.

```
procedimento GRASP_Filtro (f(.), g(.), N(.), GRASPmax, s)
1.  f* ← ∞;
2.  para (Iter = 1,2,...,Graspmax) faça
3.      Construção (g(.),α,s);
4.      se (f(s) < f*) então
5.          s* ← s;
6.          f* ← f(s);
7.      fim-se;
8.  fim-para;
9.  BuscaLocal(f(.),N(.),s);
10. s ← s*;
11. Retorne s;
fim GRASP_Filtro;
```

Figura 7: Algoritmo GRASP com Filtro

4.3.1.2 Função de Avaliação

O problema é definido como sendo de minimização, ou seja, a solução que apresentar função de avaliação de menor valor é melhor. A função de avaliação adotada foi baseada em hierarquias (com dois níveis), de forma que dois critérios podem ser avaliados. No primeiro nível é computada a soma das bandas de passagem excedentes nos enlaces sobrecarregados e no segundo nível a taxa média de ocupação de todos os enlaces da rede.

Uma função de avaliação hierárquica é composta de diversos níveis que, para comparação entre duas soluções, é considerada a melhor aquela que apresentar melhor valor no nível superior e em caso de empate recorre-se à comparação dos níveis inferiores. Assim, representando a solução do problema por uma tupla <b, t>, onde b representa a soma das bandas de passagem excedentes nos enlaces sobrecarregados e t representa a taxa média de ocupação de todos os enlaces da rede, uma solução $<b_1, t_1>$ é melhor que uma solução $<b_2, t_2>$ se a comparação lexicográfica $< b_1, t_1> < < b_2, t_2>$ for verdadeira.

4.3.1.3 Fase de Construção: First_Allocate_GRASP

Esse procedimento é utilizado para construção de uma solução inicial e é uma adaptação do método `First_Allocate` proposto em Cavalcanti [2000]. Ele recebe como parâmetros o agregado de ordem (*oa*), o grafo (*g*) e o valor alfa do procedimento GRASP (*alfa_GRASP*). Nesta fase, a solução é construída elemento a elemento, de forma iterativa. A cada iteração da fase de construção, os troncos de tráfego ainda não alocados de um mesmo agregado de ordem são colocados em uma lista de candidatos (*LC*), seguindo um critério de ordenação relativo à soma das bandas de passagem dos nós egressos. O tamanho da lista de candidatos (*LC*) é relativo ao número total de troncos de tráfego (*TT*) do agregado de ordem em questão. A partir desse número, uma lista de candidatos restrita (*LRC*) é calculada em função do parâmetro alfa, que determina o grau de gulosidade do método. No próximo passo, uma posição aleatória (*p*) da *LRC* é sorteada, para que o *TT* desta posição p seja alocado. É feita a alocação do *TT* escolhido, que passa a não pertencer mais à *LRC*, e o tamanho da *LC* é diminuído de uma unidade. O método prossegue da mesma forma, até que todos os *TTs* deste agregado de ordem tenham sido alocados.

A alocação de um tronco de tráfego é feita por um procedimento heurístico baseado em caminhos mínimos entre os nós de ingresso e egresso do fluxo e são gerados individualmente para cada dupla. Na inexistência de um caminho que atenda aos requisitos entre os nós de ingresso e egresso são consideradas relaxações no estado presente da rede, como considerar os recursos originais da topologia para se determinar uma nova alocação. No caso da topologia original de rede não comportar algum tronco de tráfego por causa dos requisitos de QoS do agregado de ordem, é gerada uma exceção, sinalizando a inviabilidade na alocação da demanda de tráfego naquela topologia de rede.

4.3.1.4 Fase de Busca Local

4.3.1.4.1 Abordagem 1: Algoritmo de Descida

O primeiro método de busca local desenvolvido foi o método de Descida, no qual todos os possíveis vizinhos da solução *s* em questão são analisados. Um vizinho candidato é aceito se representar melhora da solução que está sendo analisada. A melhora é definida pela função de avaliação de cada solução.

Foram aplicadas duas variantes do método de descida que se diferem somente pelo número de movimentos feitos na solução *s* para a construção da vizinhança *N(s)*.

a) Descida 1 Optimal

A estrutura de vizinhança escolhida foi em função dos arcos sobrecarregados de cada solução. Um vizinho é gerado otimizando-se um arco sobrecarregado da solução corrente. Dessa forma, uma solução terá tantos vizinhos quanto for o número de arcos sobrecarregados que possuir. Cada vizinho é encontrado escolhendo-se um arco sobrecarregado da solução corrente, e realocando um tronco de tráfego (*TT*) que utiliza esse enlace, visando retirar sua sobrecarga, se for possível. Para tanto, é escolhido o *TT* que utilize a menor banda de passagem do enlace, mas que retire a sobrecarga do mesmo. Caso nenhum dos *TTs* retire a sobrecarga, é escolhido o *TT* que ocupe o valor mais alto de banda de passagem do enlace.

O algoritmo de descida 1 optimal recebe como parâmetro o grafo atual, que já representa a solução inicial gerada. O critério de parada do método é em função do valor melhor_delta calculado, ser maior que zero, significando que a solução ainda pode ser melhorada e não foi atingido um ótimo local.

Para uma solução *s*, é gerado um vizinho *s'* e calculada sua *fo*. Um delta é obtido pela diferença da *fo_corrente* e da *fo_vizinho*. Caso esse delta seja melhor que o melhor_delta encontrado até então, ele é armazenado como sendo o melhor delta. Também são armazenados o *melhor_arco* (que representa o arco que deve ser otimizado), o *melhor_tt* (que representa o tronco de tráfego que deve ser realocado) e o *melhor_vertice* (que representa o vértice de saída do *melhor_arco*). Gerados todos os vizinhos de forma iterativa, é feita o movimento real de mudança de solução, a partir dos valores de melhora armazenados. Para a nova solução, é repetido o mesmo procedimento, até que não seja mais encontrada nenhuma solução de melhora.

b) Descida 2 Optimal

O algoritmo de descida 2 optimal funciona da mesma forma que o descida 1 optimal, com a diferença da estrutura de vizinhança. Na nova estrutura de vizinhança, um vizinho é gerado otimizando-se dois arcos sobrecarregados da solução corrente. Os arcos são combinados dois a dois; dessa forma, o número de vizinhos será dado por $(n^2-n)/2$, sendo n o número de arcos sobrecarregados.

4.3.1.4.2 Abordagem 2: Busca Tabu

O segundo método de busca local desenvolvido foi a metaheurística Busca Tabu [Glover, 1986], no qual é possível escapar de ótimos locais ao aceitar-se um movimento de piora. O algoritmo BT recebe o grafo *g* atual (que já representa a solução inicial gerada), o tamanho máximo da lista tabu, *tamLT,* e o número máximo de iterações sem melhora no valor da melhor solução, *BTmax,* como parâmetros. O critério de parada é decorrente de duas condições; dessa forma, o algoritmo termina caso o número de iterações sem melhora no valor da melhor solução atinja *BTmax* ou quando o valor da função de avaliação da melhor solução chega a zero, o que significa que não há inviabilidades na mesma.

A cada iteração do algoritmo é calculado o melhor vizinho *s'* da solução corrente *s*, respeitando-se a condição de que o movimento *m* que leva a tal vizinho não pertença à lista tabu (*LT*) ou embora esteja na *LT* apresente uma função de avaliação melhor do que aquela considerada a melhor até então (função de aspiração *A*). Um vizinho é gerado otimizando-se dois de seus arcos sobrecarregados da mesma forma descrita no método de descida. Após o melhor vizinho ser definido, são armazenados na *LT* os dois *TTs* que foram realocados, gerando este vizinho. Em seguida, é preciso verificar se o tamanho da *LT* está de acordo com *tamLT*; caso contrário é necessário apagar dois registros da mesma. Logo, a solução corrente s passa a ser s', não importando se sua função de avaliação é melhor ou não, o que garante ao algoritmo a possibilidade de escapar de ótimos locais. A seguir, compara-se a solução corrente com a melhor solução (*s_star*) encontrada até então e caso seja melhor, a solução *s_star* é substituída pela mesma. O algoritmo continua, de forma que uma nova iteração seja computada.

5 RESULTADOS E DISCUSSÃO

A seguir, apresentam-se os resultados computacionais obtidos a partir de testes realizados. Todos os algoritmos foram implementados na linguagem C e os testes computacionais foram realizados sob o sistema operacional Linux, em um microcomputador com processador AMD Athlon de 650 Mhz e 128 MB de memória RAM. Os parâmetros do procedimento GRASP e BT foram definidos de forma empírica a partir de testes. O número de iterações GRASP, isto é, *GRASPmax*, foi definido como 100. O valor alfa do procedimento GRASP que se mostrou mais

adequado foi 0.3. Os valores dos parâmetros *tamLT* e *BTmax* do algoritmo BT foram definidos como 6 e 10 respectivamente.

Os resultados obtidos foram baseados nos testes realizados utilizando como problema-teste, uma instância de 12 nós e 15 enlaces. Foram realizadas 30 execuções de cada método, cada qual partindo de uma semente diferente de números aleatórios. Entretanto o mesmo conjunto de sementes foi utilizado em todos os métodos. A topologia da rede está apresentada a seguir e a matriz de tráfego no Anexo A.

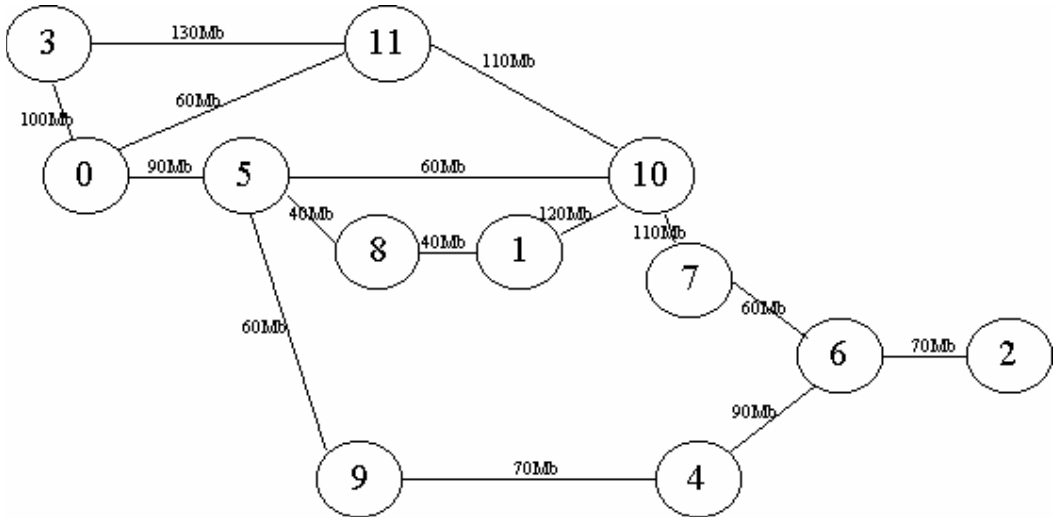


Figura 8: Topologia de rede do problema-teste

A seguir, na Tabela 1, são apresentados os melhores resultados obtidos e as médias para a aplicação de cada um dos métodos descritos. Os testes completos são apresentados no Anexo B.

Tabela comparativa 1

Método	**First_Allocate + Optimise**	**GRASP_Filtro + Optimise**		**GRASP_Filtro + Descida1Opt**		**GRASP_Filtro + Descida2Opt**		**GRASP_Filtro + Busca Tabu**	
	melhor	**melhor**	**média**	**melhor**	**média**	**melhor**	**média**	**melhor**	**média**
Tempo de CPU (s)	< 1	< 1	< 1	1	1,23	1	1,17	3	2,77
FO[1] (soma excedente nos enlaces sobrec.)	59	50	53,47	50	52,7	0	49,1	0	36,07
FO[2] (tx média de alocação em todos enlaces) (%)	68,3	63,6	67,45	63,6	67,38	55,93	66,62	49,7	62,78
Nº de enlaces sobrecarregados	5	3	4,33	3	4,33	0	4	0	3,1
Tx média de sobrealocação nos enlaces sob. (%)	15	22	19,05	22	18,68	0	17,82	0	15,36
Desvio[2] (%)	-		35,71		35,57		34,04		26,32

Na Tabela 1 são apresentados os resultados dos experimentos computacionais. A segunda coluna mostra o melhor resultado obtido a partir da aplicação dos métodos `First_Allocate + Optimise` propostos em Cavalcanti [2000], descritos nas seções 4.2.1 e 4.2.2 respectivamente. As demais colunas referem-se aos melhores resultados e às médias dos resultados obtidos com a aplicação do métodos `GRASP_Filtro + Optimise`, `GRASP_Filtro + Descida1Opt`, `GRASP_Filtro + Descida2Opt` e `GRASP_Filtro + Busca_Tabu` respectivamente.

A partir da linha 2 da Tabela 1 apresentam-se respectivamente: o tempo de execução, em segundos; os valores da função de avaliação no nível 1 (FO^1), que corresponde à soma das bandas de passagem excedentes nos enlaces sobrecarregados; os valores da função de avaliação no nível 2 (FO^2), que corresponde à taxa média de alocação em todos os enlaces; o número de enlaces sobrecarregados; a taxa média de sobrealocação nos enlaces, em porcentagem, e o desvio dos valores médios de FO^2 encontrados em relação à melhor solução obtida conforme FO^2 pela aplicação de todos os métodos, isto é:

$$\text{Desvio} = \left(\frac{FO^2 Media - FO^2 Melhor}{FO^2 Melhor} \right) \times 100$$

O método descrito em Cavalcanti [2000], denotado por `First_Allocate + Optimise`, é determinístico, ou seja, sempre que é invocado retorna o mesmo resultado para um dado problema. Dessa forma, a melhor solução encontrada para o problema-teste possui 5 enlaces sobrecarregados, com uma soma de bandas de passagem excedentes de 59 nos enlaces sobrecarregados e uma taxa de alocação média de 68,3% em todos os enlaces.

Combinando a metaheurística `GRASP_Filtro` proposta, e o algoritmo de refinamento `Optimise` [Cavalcanti, 2000], obteve-se uma solução que apresenta melhoras em relação ao método anterior, visto que a soma das bandas de passagem excedentes diminuiu para 50. Também houve melhora com relação à taxa média de alocação dos enlaces, que passou a ser 63,6%.

A melhor solução obtida pelo método proposto combinando `GRASP_Filtro + Descida1Optimal` foi a mesma encontrada por `GRASP_Filtro + Optimise`.

Através do método GRASP_Filtro + Descida2Optimal, obteve-se uma solução que não apresentou inviabilidades, isto é, não possui enlaces sobrecarregados. Além disso, a taxa média de alocação de todos os enlaces encontrada foi bem mais baixa, com valor igual a 55,93%.

O melhor resultado obtido foi pelo método GRASP_Filtro + Busca_Tabu, onde chegou-se a uma solução sem inviabilidades, com uma taxa média de alocação de todos os enlaces consideravelmente baixa, de 49,7%.

Como apresentado acima, em média, os resultados dos 30 testes realizados para cada método proposto (vide Anexo B) foram melhores do que a implementação inicial, totalmente gulosa.

A Tabela 2 apresenta a porcentagem de casos em que a aplicação de cada método mostrou-se melhor em relação ao método First_Allocate + Optimise.

Tabela comparativa 2

Método	**GRASP_Filtro + Optimise**	**GRASP_Filtro + Descida1Opt**	**GRASP_Filtro + Descida2Opt**	**GRASP_Filtro + Busca Tabu**
Nº de casos melhores (%)	93,33	93,33	96,67	100,00

A fim de garantir a robustez do algoritmo, também foram feitos testes utilizando uma topologia com 16 nodos, com graus de conectividade variáveis, gerando matrizes de tráfego para as abordagens *pipe* e *hose*. Dessa forma, é possível analisar o desempenho do algoritmo em situações variadas. O método utilizado foi GRASP_Filtro + Busca_Tabu.

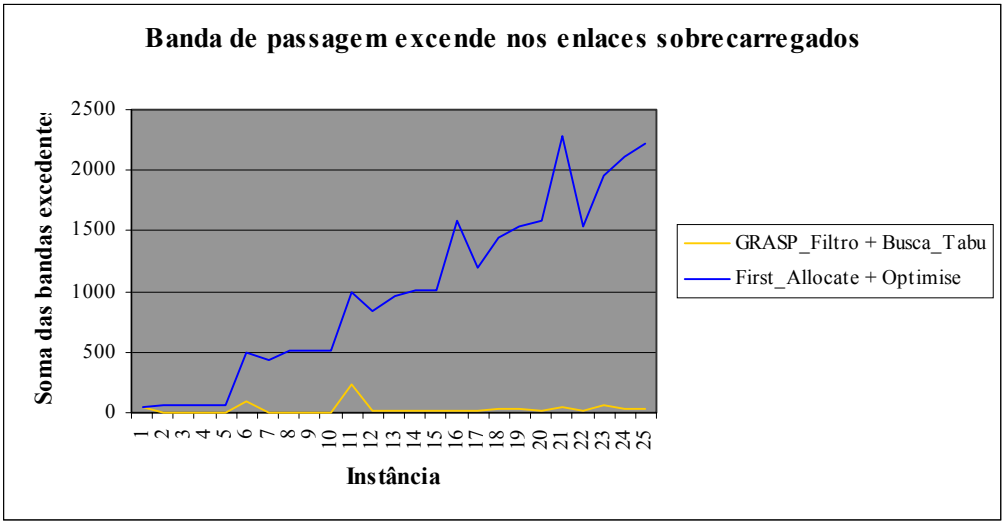
Foram testadas 100 instâncias, onde se detectou enlaces sobrecarregados em 44 delas. Comparando os resultados obtidos pelo método First_Allocate + Optimise e pelo método proposto, constatou-se que o último apresentou-se melhor em 100% dos casos, sendo que em 32 delas conseguiu remover totalmente as sobrecargas enquanto através do método First_Allocate + Optimise não foi possível.

Os gráficos apresentam os valores obtidos na FO[1] (soma das bandas de passagem excedentes nos enlaces sobrecarregados), para o método First_Allocate + Optimise e para o método GRASP_Filtro + Busca_Tabu. Cada um deles tem um grau de conectividade e refere-se a um conjunto de 25 instâncias tanto no modelo *pipe* quanto no *hose*. São alocados 240 troncos de tráfego no modelo *pipe* e 16 no

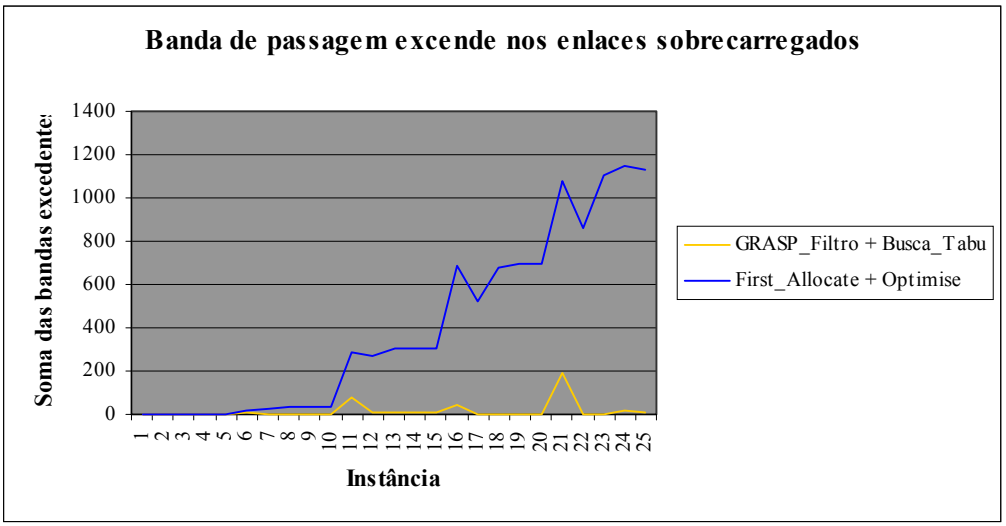
modelo *hose*, sendo o ganho de banda de passagem uma das vantagens do último modelo, já que se a taxa de ingresso for especificada em um valor maior do que o necessário para carregar o tráfego dos nós de egresso, o modelo automaticamente ajusta esse valor [Cavalcanti, 2000].

Nota-se através dos gráficos que as inviabilidades, ou enlaces sobrecarregados, ocorrem principalmente no modelo *pipe*, em função do alto número de troncos de tráfego que devem ser alocados, se comparado ao modelo *hose*.

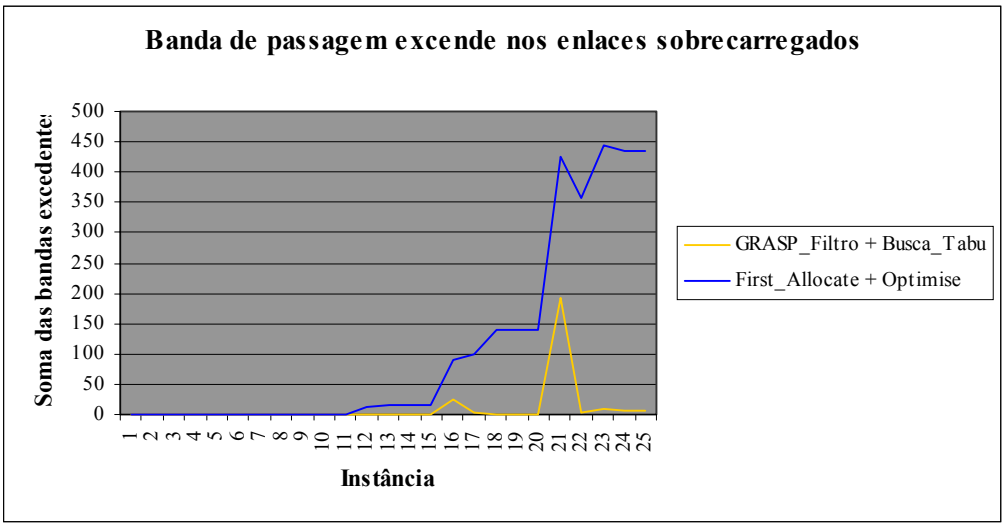
#Arcos = 70, Grau médio = 4, Capacidade original = 552.



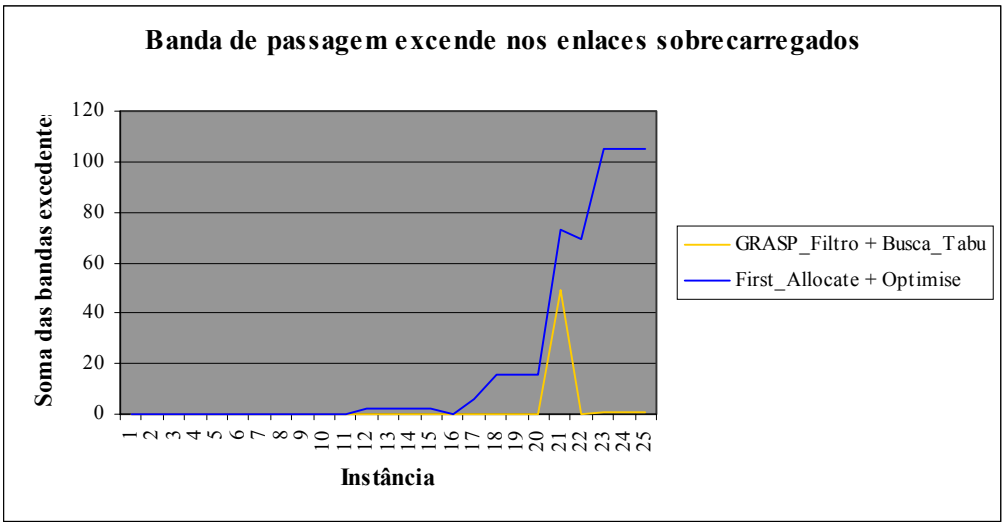
#Arcos = 98, Grau médio = 6, Capacidade original = 994.



#Arcos = 172, Grau médio = 10, Capacidade original = 1488.



#Arcos = 218, Grau médio = 13, Capacidade original = 1916.



6 CONCLUSÃO

A partir desses resultados, pode-se concluir que os métodos propostos demonstraram melhorias em relação ao método `First_Allocate + Optimise`, com redução na soma das bandas de passagem excedentes nos enlaces sobrecarregados e conseqüente redução na taxa de alocação média de todos os enlaces..

O método proposto através das metaheurísticas GRASP e Busca Tabu mostrou-se simples e eficaz na resolução do problema em questão, encontrando soluções bastante satisfatórias para o mesmo. Entretanto, é importante ressaltar que o tempo de processamento gasto, principalmente em instâncias no modelo *pipe*, é bastante elevado no método `GRASP_Filtro + Busca_Tabu` se comparado aos outros métodos.

Sendo assim, deve-se fazer uma avaliação entre os resultados obtidos e o tempo computacional gasto por este método para se saber se é ou não vantajoso o seu uso.

7 TRABALHOS FUTUROS

Aplicação de novas metaheurísticas na resolução do problema de dimensionamento de redes IP. Estudo e implementação de novas técnicas de vizinhança, que possam trazer resultados ainda mais satisfatórios.

8 REFERÊNCIAS BIBLIOGRÁFICAS

AWDUCHE, D.; MALCOLM, J.; AGOGBUA, J.; O'DELL, M. & MCMANUS, J., *"Requirements for Traffic Engineering over MPLS"*, IETF RFC 2702, 1999.

BERTSEKAS, D. & GALLAGER, R., "*Data Networks*", Prentice Hall, 1992.

BLAKE, S.; BLACK, D.; CARLSON, M.; DAVIES, E.; WANG, Z. & WEISS, W., *"An Architecture for Differentiated Services"*, IETF Informational RFC 2475, 1998.

BRADEN, R.; CLARK, D. & SHENKER, S., *Integrated Services in the Internet Architecture: an Overview*, Internet RFC 1633, 1994.

CAVALCANTI, C.F.M.C., "*Algoritmo de Dimensionamento de Rede"*, Tese de Doutorado, DCC/UFMG, 2000.

FEO, T.A. & RESENDE, M.G.C., "*Greedy randomized adaptive search procedures",* Journal of Global Optimization, 6:109-133, 1995.

FERGUSON, P. & HUSTON, G., *Quality of Service in the Internet: Fact, Fiction, or Compromise?,* INET'98, Julho 1998.

GLOVER, F. *Future paths for Integer Programming and links to Artificial Intelligence*. Computers and Operations Research, 5:553-549, 1986.

GODERIS, D. et al., *"Functional Architecture Definition and Top Level Design"*, Technical Report Tequila Project – Deliverable D1.1, 2000. Disponível em: http://www.ist-tequila.org/deliverables/D1-1.pdf

GODERIS, D. et al., *"Overview and Principles of Internet Traffic Engineering"*, IETF Internet Draft: RFC 3272, 2001. Disponível em: http://www.ist-tequila.org.

HANSEN, P. *The steepest ascent mildest descent heuristic for combinatorial programming*. In Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, 1986.

HEINAMEN, J. et al., *"Assured Forwarding PHB Group"*, RFC 2597, Junho 1999.

JACOBSON, V.; NICHOLS, K. & PODURI, K., *"An Expedited Forwarding PHB"*, RFC 2598, Junho 1999.

KAMIENSKI, C.A., *Qualidade de Serviço na Internet*, Universidade Federal de Pernambuco, 1999.
Disponível em: www.cin.ufpe.br/~cak/publications/ kamienski-qos-eine-99.pdf

KUMAR, A. et al., *"Algorithms for Provisioning Virtual Private Networks in the Hose Model"*, In Proceedings ACM SIGCOMM'01 Conference, 2001.

MLADENOVIC, N. & HANSEN, P., *"Variable Neighborhood Search"*, Computers and Operations Research, v. 24, p. 1097 – 1100, 1997.

PORTNOI, M. & ARAÚJO, R.G.B., *Network Simulator – Visão Geral da Ferramenta de Simulação de Redes*, 2002.
Disponível em: http://locksmith.orcishweb.com/articles/networksimulator-sepa.pdf

RABBAT, R.; LABERTEAUX, K.; MODI, N. & KENNEY, J., "*Traffic Engineering Algorithms Using MPLS for Service Differentiation*". International Conference on Communications IEEE ICC'2000, New Orleans, Junho 2000.

ROSEN, E. & CALLON, R., *"Multiprotocol Label Switching Architeture"*, RFC 3031, 2001. Disponível em: www.ietf.org/rfc/rfc3031.txt

SOUZA, M.J.F., "*Notas de aula da disciplina Inteligência Computacional para Otimização"*. Disponível em: www.decom.ufop.br/prof/marcone.

TEITELBAUM, B., *"Qbone Architecture (v1.0)"*, Internet2 QoS Working Group draft.
Disponível em: www.internet2.edu/qos/wg/papers/qbArch/1.0/draft-i2-qbone-arch-1.0.html

TEITELMAN, B. & HANSS, T., *QoS Requirements for Internet2*, Internet2 QoS Work Group Draft, Abril 1998.

TRIMINTZIOS, P. et al., *"A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks"*, IEEE Communications Magazine, vol. 39, no. 5, 2001.

ANEXO A

Troncos de Tráfego do problema-teste:

#				in	bw	eg	bw				
#id	in	eg	OA	min	max	min	max	delay	loss	start	end
0	0	11	EF	20	20	20	20	10.0	0.1	0	0
1	0	11	EF	30	30	30	30	10.0	0.1	0	0
2	0	11	EF	40	40	40	40	10.0	0.1	0	0
3	0	11	EF	50	50	50	50	10.0	0.1	0	0
4	0	3	EF	20	20	20	20	10.0	0.1	0	0
5	3	11	EF	54	54	54	54	10.0	0.1	0	0
6	0	3	EF	53	53	33	33	10.0	0.1	0	0
6	0	11	EF	53	53	23	23	10.0	0.1	0	0
7	0	3	EF	52	52	52	52	10.0	0.1	0	0
7	0	11	EF	52	52	22	22	10.0	0.1	0	0
8	5	10	EF	55	55	35	35	10.0	0.1	0	0
8	5	11	EF	55	55	25	25	10.0	0.1	0	0
9	0	1	EF	20	20	1	1	10.0	0.1	0	0
9	0	2	EF	20	20	1	1	10.0	0.1	0	0
9	0	3	EF	20	20	1	1	10.0	0.1	0	0
9	0	4	EF	20	20	1	1	10.0	0.1	0	0
9	0	5	EF	20	20	1	1	10.0	0.1	0	0
9	0	6	EF	20	20	1	1	10.0	0.1	0	0
9	0	7	EF	20	20	1	1	10.0	0.1	0	0
9	0	8	EF	20	20	1	1	10.0	0.1	0	0
9	0	9	EF	20	20	1	1	10.0	0.1	0	0
9	0	10	EF	20	20	1	1	10.0	0.1	0	0
10	1	0	EF	20	20	1	1	10.0	0.1	0	0
10	1	2	EF	20	20	1	1	10.0	0.1	0	0
10	1	3	EF	20	20	1	1	10.0	0.1	0	0
10	1	4	EF	20	20	1	1	10.0	0.1	0	0
10	1	5	EF	20	20	1	1	10.0	0.1	0	0
10	1	6	EF	20	20	1	1	10.0	0.1	0	0
10	1	7	EF	20	20	1	1	10.0	0.1	0	0
10	1	8	EF	20	20	1	1	10.0	0.1	0	0
10	1	9	EF	20	20	1	1	10.0	0.1	0	0
10	1	10	EF	20	20	1	1	10.0	0.1	0	0
11	2	0	EF	20	20	1	1	10.0	0.1	0	0
11	2	1	EF	20	20	1	1	10.0	0.1	0	0
11	2	3	EF	20	20	1	1	10.0	0.1	0	0
11	2	4	EF	20	20	1	1	10.0	0.1	0	0
11	2	5	EF	20	20	1	1	10.0	0.1	0	0
11	2	6	EF	20	20	1	1	10.0	0.1	0	0
11	2	7	EF	20	20	1	1	10.0	0.1	0	0
11	2	8	EF	20	20	1	1	10.0	0.1	0	0
11	2	9	EF	20	20	1	1	10.0	0.1	0	0
11	2	10	EF	20	20	1	1	10.0	0.1	0	0
12	3	0	EF	20	20	1	1	10.0	0.1	0	0
12	3	1	EF	20	20	1	1	10.0	0.1	0	0
12	3	2	EF	20	20	1	1	10.0	0.1	0	0
12	3	4	EF	20	20	1	1	10.0	0.1	0	0
12	3	5	EF	20	20	1	1	10.0	0.1	0	0
12	3	6	EF	20	20	1	1	10.0	0.1	0	0
12	3	7	EF	20	20	1	1	10.0	0.1	0	0
12	3	8	EF	20	20	1	1	10.0	0.1	0	0
12	3	9	EF	20	20	1	1	10.0	0.1	0	0
12	3	10	EF	20	20	1	1	10.0	0.1	0	0
13	4	0	EF	20	20	1	1	10.0	0.1	0	0
13	4	1	EF	20	20	1	1	10.0	0.1	0	0
13	4	2	EF	20	20	1	1	10.0	0.1	0	0
13	4	3	EF	20	20	1	1	10.0	0.1	0	0
13	4	5	EF	20	20	1	1	10.0	0.1	0	0
13	4	6	EF	20	20	1	1	10.0	0.1	0	0
13	4	7	EF	20	20	1	1	10.0	0.1	0	0
13	4	8	EF	20	20	1	1	10.0	0.1	0	0
13	4	9	EF	20	20	1	1	10.0	0.1	0	0
13	4	10	EF	20	20	1	1	10.0	0.1	0	0
14	4	0	EF	20	20	1	1	10.0	0.1	0	0
14	4	1	EF	20	20	1	1	10.0	0.1	0	0
14	4	2	EF	20	20	1	1	10.0	0.1	0	0
14	4	3	EF	20	20	1	1	10.0	0.1	0	0
14	4	5	EF	20	20	1	1	10.0	0.1	0	0
14	4	6	EF	20	20	1	1	10.0	0.1	0	0
14	4	7	EF	20	20	1	1	10.0	0.1	0	0

14	4	8	EF	20	20	1	1	10.0	0.1	0	0
14	4	9	EF	20	20	1	1	10.0	0.1	0	0
14	4	10	EF	20	20	1	1	10.0	0.1	0	0
15	5	0	EF	20	20	1	1	10.0	0.1	0	0
15	5	1	EF	20	20	1	1	10.0	0.1	0	0
15	5	2	EF	20	20	1	1	10.0	0.1	0	0
15	5	3	EF	20	20	1	1	10.0	0.1	0	0
15	5	4	EF	20	20	1	1	10.0	0.1	0	0
15	5	6	EF	20	20	1	1	10.0	0.1	0	0
15	5	7	EF	20	20	1	1	10.0	0.1	0	0
15	5	8	EF	20	20	1	1	10.0	0.1	0	0
15	5	9	EF	20	20	1	1	10.0	0.1	0	0
15	5	10	EF	20	20	1	1	10.0	0.1	0	0
16	0	1	EF	20	20	1	1	10.0	0.1	0	0
16	0	2	EF	20	20	1	1	10.0	0.1	0	0
16	0	3	EF	20	20	1	1	10.0	0.1	0	0
16	0	4	EF	20	20	1	1	10.0	0.1	0	0
16	0	5	EF	20	20	1	1	10.0	0.1	0	0
16	0	6	EF	20	20	1	1	10.0	0.1	0	0
16	0	7	EF	20	20	1	1	10.0	0.1	0	0
16	0	8	EF	20	20	1	1	10.0	0.1	0	0
16	0	9	EF	20	20	1	1	10.0	0.1	0	0
16	0	10	EF	20	20	1	1	10.0	0.1	0	0
17	1	0	EF	20	20	1	1	10.0	0.1	0	0
17	1	2	EF	20	20	1	1	10.0	0.1	0	0
17	1	3	EF	20	20	1	1	10.0	0.1	0	0
17	1	4	EF	20	20	1	1	10.0	0.1	0	0
17	1	5	EF	20	20	1	1	10.0	0.1	0	0
17	1	6	EF	20	20	1	1	10.0	0.1	0	0
17	1	7	EF	20	20	1	1	10.0	0.1	0	0
17	1	8	EF	20	20	1	1	10.0	0.1	0	0
17	1	9	EF	20	20	1	1	10.0	0.1	0	0
17	1	10	EF	20	20	1	1	10.0	0.1	0	0
18	2	0	EF	20	20	1	1	10.0	0.1	0	0
18	2	1	EF	20	20	1	1	10.0	0.1	0	0
18	2	3	EF	20	20	1	1	10.0	0.1	0	0
18	2	4	EF	20	20	1	1	10.0	0.1	0	0
18	2	5	EF	20	20	1	1	10.0	0.1	0	0
18	2	6	EF	20	20	1	1	10.0	0.1	0	0
18	2	7	EF	20	20	1	1	10.0	0.1	0	0
18	2	8	EF	20	20	1	1	10.0	0.1	0	0
18	2	9	EF	20	20	1	1	10.0	0.1	0	0
18	2	10	EF	20	20	1	1	10.0	0.1	0	0
19	3	0	EF	20	20	1	1	10.0	0.1	0	0
19	3	1	EF	20	20	1	1	10.0	0.1	0	0
19	3	2	EF	20	20	1	1	10.0	0.1	0	0
19	3	4	EF	20	20	1	1	10.0	0.1	0	0
19	3	5	EF	20	20	1	1	10.0	0.1	0	0
19	3	6	EF	20	20	1	1	10.0	0.1	0	0
19	3	7	EF	20	20	1	1	10.0	0.1	0	0
19	3	8	EF	20	20	1	1	10.0	0.1	0	0
19	3	9	EF	20	20	1	1	10.0	0.1	0	0
19	3	10	EF	20	20	1	1	10.0	0.1	0	0
20	4	1	EF	20	20	1	1	10.0	0.1	0	0
20	4	2	EF	20	20	1	1	10.0	0.1	0	0
20	4	3	EF	20	20	1	1	10.0	0.1	0	0
20	4	5	EF	20	20	1	1	10.0	0.1	0	0
20	4	6	EF	20	20	1	1	10.0	0.1	0	0
20	4	7	EF	20	20	1	1	10.0	0.1	0	0
20	4	8	EF	20	20	1	1	10.0	0.1	0	0
20	4	9	EF	20	20	1	1	10.0	0.1	0	0
20	4	10	EF	20	20	1	1	10.0	0.1	0	0
21	4	1	EF	20	20	1	1	10.0	0.1	0	0
21	4	2	EF	20	20	1	1	10.0	0.1	0	0
21	4	3	EF	20	20	1	1	10.0	0.1	0	0
21	4	5	EF	20	20	1	1	10.0	0.1	0	0
21	4	6	EF	20	20	1	1	10.0	0.1	0	0
21	4	7	EF	20	20	1	1	10.0	0.1	0	0
21	4	8	EF	20	20	1	1	10.0	0.1	0	0
21	4	9	EF	20	20	1	1	10.0	0.1	0	0
21	4	10	EF	20	20	1	1	10.0	0.1	0	0
22	5	0	EF	20	20	1	1	10.0	0.1	0	0
22	5	1	EF	20	20	1	1	10.0	0.1	0	0
22	5	2	EF	20	20	1	1	10.0	0.1	0	0
22	5	3	EF	20	20	1	1	10.0	0.1	0	0
22	5	4	EF	20	20	1	1	10.0	0.1	0	0
22	5	6	EF	20	20	1	1	10.0	0.1	0	0

22	5	7	EF	20	20	1	1	10.0	0.1	0	0
22	5	8	EF	20	20	1	1	10.0	0.1	0	0
22	5	9	EF	20	20	1	1	10.0	0.1	0	0
22	5	10	EF	20	20	1	1	10.0	0.1	0	0

ANEXO B

Tabelas de resultados do problema-teste

First Allocate + Optimise

Teste	Tempo de CPU (s)	Soma excedente	Nº enlaces sob.	Tx média alocação nos enlaces sob. (%)	Tx média alocação em todos os enlaces (%)
1	**< 1**	**59**	**5**	**115**	**68,3**

GRASP_Filtro com Optimise

Teste	Tempo de CPU	Soma excedente[1]	Nº enlaces sob.	Tx méd sobrealocação nos enlaces sob. (%)	Tx média alocação em todos os enlaces (%)[2]
1	< 1	56	6	14,67	72
2	< 1	62	5	17	68,1
3	< 1	53	4	20,75	67,83
4	< 1	50	5	15,8	68,63
5	< 1	56	6	14,67	72
6	< 1	50	4	20,25	65,9
7	< 1	54	5	14,2	68,3
8	< 1	52	4	17,5	67,1
9	< 1	56	5	17,8	68,43
10	< 1	50	3	22	63,6
11	< 1	50	5	13,2	68,73
12	< 1	50	4	20	65,67
13	< 1	61	6	11,5	70,53
14	< 1	50	4	16,5	65,73
15	< 1	58	4	21,75	68,93
16	< 1	54	3	26,67	64
17	< 1	51	4	20,75	67,73
18	< 1	52	5	13,6	69
19	< 1	55	4	22,25	66,93
20	< 1	56	3	30,33	64,03
21	< 1	54	3	30	64,33
22	< 1	50	5	13	68,73
23	< 1	56	6	14,67	72
24	< 1	51	3	27	65,23
25	< 1	50	4	16,5	67,2
26	< 1	54	4	20	67,4
27	< 1	55	4	18,75	67
28	< 1	56	5	17,8	68,23
29	< 1	52	4	20,5	66,63
30	< 1	50	3	22	63,6
Média	**< 1**	**53,47**	**4,33**	**19,05**	**67,45**

Desvio: (fo_medio – fo_star)/fo_star*100

Desvio[2] = (67,45 – 49,7)/49,7*100 = 35,71

GRASP_Filtro com Descida1Opt

Teste	Tempo de CPU	Soma excedente[1]	Nº enlaces sob.	Tx méd sobrealocação nos enlaces sob. (%)	Tx média alocação em todos os enlaces (%)[2]
1	1	55	5	17,6	68,4
2	1	61	6	13,83	71,57
3	2	51	4	20	67,6
4	2	50	5	15,8	68,63
5	1	55	5	17,6	68,4
6	1	50	4	19,25	66,37
7	1	54	5	14,4	68,5
8	1	52	4	17,5	67,1
9	1	56	5	17,8	68,43
10	1	50	3	22	63,6
11	2	50	5	13,2	68,73
12	1	50	4	20	65,67
13	1	61	6	12	71,5
14	1	50	4	16,5	65,73
15	1	57	6	15	72,3
16	1	54	3	26,33	64,4
17	1	51	4	21	67,77
18	2	52	5	13,6	69
19	2	50	4	19,25	66,53
20	1	53	3	29	64,9
21	1	53	3	29	64,53
22	1	50	5	13	68,73
23	1	55	5	17,6	68,4
24	1	51	3	27	65,23
25	1	50	4	16,5	67,2
26	2	54	4	20,5	67,43
27	2	50	4	16	66,63
28	1	56	5	17,8	68,23
29	1	50	4	19,5	66,3
30	1	50	3	22	63,6
Média	**1,23**	**52,7**	**4,33**	**18,68**	**67,38**

Desvio: (fo_medio – fo_star)/fo_star*100

Desvio² = (67,38 – 49,7)/49,7*100 = 35,57

GRASP_Filtro com Descida2Opt

Teste	Tempo de CPU	Soma excedente[1]	Nº enlaces sob.	Tx méd sobrealocação nos enlaces sob. (%)	Tx média alocação em todos os enlaces (%)[2]
1	1	55	5	17,6	68,4
2	1	0	0	0	55,93
3	1	51	4	20	67,6
4	1	50	5	15,8	68,63
5	2	55	5	17,6	68,4
6	1	50	4	19,25	66,37
7	1	0	0	0	57,43
8	1	52	4	17,5	67,1
9	1	56	5	17,8	68,43
10	1	50	3	22	63,6
11	1	53	6	11,5	72,4
12	2	50	4	20	65,67
13	1	61	6	12	71,5
14	1	50	4	16,5	65,73
15	1	54	4	22,25	65,73
16	2	54	3	26,33	64,4
17	1	51	4	21	67,77
18	1	52	5	13,6	69
19	1	50	4	19,25	66,53
20	2	53	3	29	64,9
21	1	53	3	29	64,53
22	1	53	6	11,33	72,4
23	1	55	5	17,6	68,4
24	1	51	3	27	65,23
25	1	50	4	16,5	67,2
26	1	54	4	20,5	67,43
27	1	54	5	14,4	69,87
28	2	56	5	17,8	68,23
29	1	50	4	19,5	66,3
30	1	50	3	22	63,6
Média	**1,17**	**49,1**	**4**	**17,82**	**66,62**

Desvio: (fo_medio – fo_star)/fo_star*100

Desvio² = (66,62 – 49,7)/49,7*100 = 34,04

GRASP_Filtro com Busca Tabu

Teste	Tempo de CPU	Soma excedente[1]	Nº enlaces sob.	Tx méd sobrealocação nos enlaces sob. (%)	Tx média alocação em todos os enlaces (%)[2]
1	3	52	5	14,2	67,33
2	2	0	0	0	51,63
3	3	50	4	20,25	66,83
4	3	45	5	14,2	70,3
5	2	52	5	14,6	67,83
6	3	40	3	20,33	63,9
7	4	0	0	0	50,37
8	3	3	2	3	57,53
9	3	52	5	14,8	68,27
10	3	45	2	29	60,73
11	3	45	4	18	66,27
12	3	2	1	2	52,07
13	3	0	0	0	49,7
14	3	43	3	23,33	62,67
15	3	16	2	14	59,77
16	3	50	4	20,25	67,43
17	3	35	2	20,5	60,43
18	3	40	4	16,25	66,63
19	3	50	4	19,25	66,53
20	3	53	3	29	64,9
21	3	53	3	29	64,53
22	2	52	5	14,2	67,33
23	2	52	5	14,6	67,83
24	3	51	3	27	65,23
25	2	50	4	16,5	67,2
26	2	10	1	16	55
27	2	1	1	1	52,77
28	2	45	5	13,4	68,87
29	3	45	5	14	70,03
30	3	50	3	22	63,6
Média	**2,77**	**36,07**	**3,10**	**15,36**	**62,78**

Desvio: (fo_medio – fo_star)/fo_star*100

Desvio² = (62,78 – 49,7)/49,7*100 = 26,32

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO - DECOM
CAMPUS UNIVERSITÁRIO - MORRO DO CRUZEIRO
CEP 35400-000 - OURO PRETO - MINAS GERAIS