# UM GRASP PARA O PROBLEMA DE ATRIBUIÇÃO DE LOCALIDADES A ANÉIS SONET

**Lucas de Oliveira Bastos**
Instituto de Computação, Universidade Federal Fluminense
Endereço: Rua Passo da Pátria, 156, 24210-240, Niterói – RJ – Brasil
E-mail: lbastos@ic.uff.br

**Luiz Satoru Ochi**
Instituto de Computação, Universidade Federal Fluminense
Endereço: Rua Passo da Pátria, 156, 24210-240, Niterói – RJ – Brasil
E-mail: satoru@ic.uff.br

**Elder M. Macambira**
Departamento de Estatística, Universidade Federal da Paraíba
Endereço: Cidade Universitária, 58051-900, João Pessoa – PB – Brasil
E-mail: elder@de.ufpb.br

**Resumo**

Trataremos do problema de atribuição de localidades a anéis SONET, um problema de otimização combinatória presente no projeto de redes de telecomunicações. Neste problema, cada cliente deve se conectar à exatamente um anel SONET enquanto um anel especial interconecta os demais anéis entre si. É também imposta uma restrição de capacidade sobre cada anel. O problema se resume a encontrar uma atribuição viável dos clientes minimizando o número total de anéis utilizados. Descreveremos um procedimento de busca guloso adaptativo randomizado (GRASP) para o problema. O procedimento é baseado num conjunto de algoritmos construtivos e no uso de uma função objetivo variável. Mostraremos evidências empíricas extensivas da eficiência do nosso algoritmo na prática. Os resultados indicam que a implementação do GRASP proposta se compara favoravelmente aos métodos gulosos e implementações de busca tabu, *scatter search* e *path-relinking* previamente propostos para o problema. Para a maioria das instâncias de teste encontradas na literatura, encontramos as soluções ótimas ou outras muito próximas ao ótimo.

**palavras-chave** Problema de Atribuição de Localidades a Anéis SONET, Particionamento de Grafos, Meta heurísticas, GRASP.

**Abstract**

We consider the SONET ring assignment problem, a combinatorial optimization problem that arises in telecommunications networks design. In this problem, each customer has to be assigned to exactly one SONET ring and a special ring interconnects the others rings together. A capacity constraint on each ring is also imposed. The problem is to find a feasible assignment of the customers minimizing the total number of rings used. We describe a greedy randomized adaptive search procedure (GRASP) for the SONET ring assignment problem. The procedure is based on a set of constructive algorithms and the use of a variable objective function. We show extensive empirical evidences to the effectiveness of our algorithm in practice. These results indicate that the proposed GRASP implementation compares favorably to greedy methods and implementations of tabu search, scatter search and path relinking previously proposed for the problem. For most benchmarks instances in the literature, we obtained solutions that are either optimal or very close to it.

**keywords** SONET Ring Assignment Problem, Graph Partitioning, Metaheuristics, GRASP.

## 1 Introdução

Considere o seguinte problema de partição por vértice. Seja $G = (V, E)$ um grafo não direcionado e um inteiro positivo $B$. Associado a cada aresta $(u, v)$ em $E$ há um inteiro não negativo $d_{uv}$. Uma solução viável do problema é uma partição dos vértices em $G$ em conjuntos disjuntos $V_1, V_2, ..., V_k$ tais que:

$$(i) \sum_{(u,v) \in G[V_j]} d_{uv} \leq B \text{ , para todo } j \in \{1,\ldots,k\}, \text{ e} \quad (1)$$

$$(ii) \sum_{j=1}^{k} \sum_{(u,v) \in \delta(V_j) \mid u<v} d_{uv} \leq B \text{ ,} \quad (2)$$

onde $G[V_j]$ é o grafo induzido por $V_j$ em $G$ e $\delta(V_j)$ é o conjunto de arestas com exatamente uma extremidade em $V_j$. A meta é encontrar uma solução viável que minimize o tamanho da partição, isto é, o valor de $k$.

Este problema de particionamento de grafo é também conhecido como o problema de atribuição de localidades a anéis SONET (ou SDH), ou simplesmente SRAP (SONET Ring Assignment Problem). O SRAP foi investigado recentemente por Goldschmidt *et al.* em [3]. Em seu artigo os autores motivaram o estudo do problema mostrando sua relevância para o projeto de redes de telecomunicações de fibra-ótica utilizando a tecnologia SONET (Synchronous Optical NETwork) ou SDH (Synchronous Digital Hierarchy).

Neste contexto, os vértices do grafo $G = (V,E)$, com $|V| = n$, são associados às localidades clientes enquanto as arestas são associadas à existência de demandas de tráfico entre pares de localidades. Os pesos das arestas medem a demanda de comunicação atual entre clientes. Dada uma solução viável para o problema de particionamento de grafos introduzido acima, os vértices em cada subconjunto da partição formam um *anel local* ou simplesmente um *anel*. Devido às limitações físicas sobre os equipamentos utilizados na construção da rede, a demanda total em cada anel não pode exceder uma constante $B$. Além disso, a demanda total entre clientes em diferentes clusters é gerenciada por um anel adicional chamado de *anel federal*. A conexão entre um anel local e o anel federal se dá por meio de um dispositivo conhecido como *digital cross-connect system* (DCS). Uma vez que a capacidade do anel federal é também limitada por B, a soma dos pesos das arestas no multicorte correspondente a qualquer solução viável não pode ser maior que esta quantidade. Finalmente, como os DCS's são de longe os equipamentos mais caros necessários para a implementação de uma rede, um problema básico no projeto de redes SONET de baixo custo requer uma solução que minimize o número de anéis locais. É fácil verificar que o problema de partição de grafos discutido no início desta seção modela corretamente este problema.

Veja Goldschmidt *et al.* [3] para uma discussão mais detalhada sobre a arquitetura de redes SONET e uma revisão da literatura sobre o SRAP. Em seu trabalho os autores provaram que o SRAP é NP-difícil. Eles também propuseram três diferentes heurísticas gulosas para resolvê-lo. Um outro trabalho sobre heurísticas para o SRAP pode ser encontrado em [1] onde são propostas meta-heurísticas *tabu search, path relinking* e *scatter search*. Recentemente, Macambira [4] propôs formulações de programação inteira e métodos exatos para resolver o SRAP.

Este trabalho está focado na solução do SRAP via meta-heurísticas. Apresentamos um GRASP para este problema, que faz uso de uma estrutura de vizinhança relativa. Na Seção 2, descrevemos as fases de construção e busca local na seção 3, apresentamos os resultados computacionais para as instâncias de teste do problema. Observações e conclusões são feitas na Seção 4.

## 2 Heurística GRASP

GRASP [2] é um procedimento de inícios múltiplos, onde pontos diferentes do espaço de busca são investigados pela busca local na tentativa de encontrar soluções de alta qualidade. Cada iteração do GRASP consiste na construção de uma solução de forma gulosa aleatória, seguida pela aplicação de

busca local sobre a solução construída. A melhor solução encontrada em todas as iterações é retornada como o resultado.

No resto dessa seção, descreveremos em detalhe as fases do GRASP para o SRAP, ou seja, as fases de construção e busca local. Para descrever a fase de construção é necessário prover uma definição candidata (para a lista restrita de candidatos) e uma função gulosa adaptativa, e especificar o mecanismo de restrição de candidatos. Para a fase de busca local, é necessário definir a vizinhança de busca e especificar o algoritmo de busca local.

## 2.1 Fase de Construção

Note que é possível que o GRASP proposto retorne soluções inviáveis. Naturalmente isto deve ser permitido já que, às vezes, não é possível decidir em tempo computacional razoável se a instância possui solução ou não. Mesmo quando a fase construtiva descobre que a instância é viável, soluções inviáveis podem ser visitadas.

A meta, nesse caso, é permitir maior flexibilidade de movimento através do espaço de busca. Contudo a função objetivo deve ser modificada para tornar as soluções viáveis mais atrativas do que as inviáveis. Para explicar as mudanças na função objetivo, definimos o seguinte valor associado a uma solução viável $S$ com conjunto de vértices $V_1,...,V_r$:

$$BN = \max\left(0, \sum_{j=1}^{r} \sum_{(u,v)\in\delta(V_j)\,|\,u<v} d_{uv}\right), \tag{3}$$

onde $BN$ indica o tráfico total através do anel federal. O custo desta solução na função objetivo modificada é computado pela fórmula:

$$z = rB \text{ (solução viável) ou,} \tag{4}$$

$$z = 2r + BN \text{ (solução inviável).} \tag{5}$$

Descreveremos agora a fase de construção do GRASP. O algoritmo é uma adaptação das três heurísticas gulosas apresentadas em [3] para incluir randomização. Inicialmente, é gerada uma solução com $n$ anéis com um vértice atribuído a cada anel. Estes anéis são viáveis desde que a instância seja viável. Dessa forma, a única restrição que pode ser violada é a que impõe um limite sobre a demanda no anel federal.

A cada iteração, a união de dois anéis em um único anel é considerada. Tal união é permitida apenas se o anel resultante for viável. Claramente, esta operação reduz a quantidade de demanda no anel federal. A escolha gulosa é guiada pelas demandas nas arestas que estão no multicorte dos anéis. A randomização não nos forçará a pegar a melhor aresta (heurística *edge-based*), melhor corte (heurística *cut-based*) ou o melhor nó (heurística *node-based*). Ao invés disso, a aresta, corte ou nó selecionado (a) é escolhido (a) aleatoriamente entre os melhores candidatos na lista restrita de candidatos com as maiores demandas.

Desenvolvemos ainda um novo algoritmo para a fase de construção tentando aperfeiçoar os resultados gerados pelas heurísticas gulosas. Esta nova proposta é descrita a seguir.

O objetivo da Heurística de Vizinhança Relativa (HVR), aqui introduzida, é estabelecer conjuntos de localidades vizinhas em relação a um certo critério para serem mantidas nos mesmos anéis. O critério considerado aqui estabelece que duas localidades são consideradas vizinhas relativas se compartilham diretamente a maior demanda entre todas as suas demandas comuns. Após esse passo, todas as localidades vizinhas são reunidas num anel. Este procedimento gera no mínimo um anel e no máximo $(n/2)$ anéis, sendo $n$ o número de localidades do problema. Localidades restantes são colocadas num anel com probabilidade dada por $p = f * dem(i, a)$ onde $f$ é um fator aleatório e $dem(i, a)$ é a demanda total entre a localidade $i$ e o anel $a$. Depois disso, a HVR tenta unir anéis da mesma forma como a heurística *cut-based* o faz. Veja Goldschmidt *et al.* [3] para mais detalhes sobre a heurística *cut-based*.

Os passos do algoritmo de construção HVR se encontram resumidos na Figura 1.

```
procedimento HVR( )
01 //-- Gerando informações sobre vizinhanças:
02 para cada aresta (i, j) faça
03   se peso(i, j) >= max(arestaMaiorPeso(i), arestaMaiorPeso (j))
04     definirVizinhos(i, j);
05 //-- Gerando anéis a partir das informações de vizinhança:
06 para cada par (i, j) de localidades faça
07   se sãoVizinhos(i, j)
08     juntarNumAnel(i, j);
09 //-- Alocando localidades restantes:
10 para cada i = próximaLocalidadeDesalocada( ) faça
11   porNumAnel(i);
12 //-- Unindo anéis:
13 para cada par (i, j) de anéis faça
14   se podemSeUnir (i, j)
15     unirAneis(i, j);
fim do procedimento
```

**Figura 1: Pseudo-código da Heurística de Vizinhança Relativa.**

## 2.2 Fase de Busca Local

Depois que uma solução é construída, uma fase de busca local deve ser executada na tentativa de melhorar a solução inicial. A fase de busca local é baseada em dois tipos de vizinhança.

A primeira vizinhança considera a possibilidade de mover cada vértice *u* de seu anel para quaisquer dos outros anéis existentes. A solução base inicial para a busca local é obtida da fase construtiva. Cada vértice *u* é movido do seu anel *s* para o anel *t*. Observamos que a movimentação do nó *u* de *s* para *t* modifica os valores de tráfego e uma nova solução *z(.)* é computada. Se para cada tripla *(u,s,t)*, o valor de *z(.)* tiver sido melhorado, então a troca é realizada, e uma nova solução base é criada.

```
procedimento BuscaLocal( )
01 //-- Aplicando Primeira Vizinhança:
02 faça {
03   para cada vértice pertencente ao menor anel faça
04     enviarParaOMelhorAnelQuePodeComportá-Lo( );
05 } até não poder mais mover um nó;
06 //-- Aplicando Segunda Vizinhança:
07 para cada anel i faça
08   para cada anel  j faça
09     para cada nó n pertencente a i faça {
10       tentarTrocar(n, nósDe( j));
11       se soluçãoMelhorou ( )
12         reinicieBuscaLocal( );
13     }
fim procedimento
```

**Figura 2: Pseudo-código do procedimento de Busca Local.**

A segunda vizinhança considera todas as possibilidades de trocar pares de vértices entre diferentes anéis. Se nenhuma troca melhorar o valor de *z(.)*, então a busca local é finalizada. Caso contrário, depois que os elementos que levam ao menor valor de *z(.)* são trocados, uma nova solução base é criada e a busca local é realizada novamente. As vizinhanças são inspecionadas na ordem sugerida pelo algoritmo na Figura 2.

## 3 Resultados Computacionais

Nesta seção, ilustraremos os resultados da aplicação do nosso algoritmo sobre problemas testes gerados aleatoriamente. Todos os testes foram executados num computador pessoal equipado com um processador Pentium IV 1700 MHz e 256 Mbytes de memória RAM rodando o sistema operacional Linux. O código fonte do GRASP foi escrito em linguagem C++.

Antes de descrever os resultados experimentais, devemos comentar a respeito das instâncias utilizadas nos testes. O conjunto de instâncias foi dividido em duas categorias. As instâncias na categoria C1 foram tiradas de [3] e correspondem às instâncias tipo 1 e 2 naquele artigo. A categoria C2 é composta por todas as instâncias testadas em [1] excluindo aquelas já em C1.

No total, há 111 instâncias em C1 e 230 em C2. Dentro de cada categoria, as instâncias estão divididas em geométricas e randômicas e, então, depois divididas naquelas que contém anéis de capacidades baixa e alta, respectivamente, 155 Mbs e 622 Mbs. Estas características de uma instância podem ser deduzidas de seus nomes. Nomes de instâncias sempre iniciam com duas letras. A primeira letra é sempre “G” ou “R” significando que as instâncias pertencem à subdivisão geométrica e randômica, respectivamente. A segunda letra é “L” ou “H”, dependendo da capacidade dos anéis ser 155 Mbs ou 622 Mbs, respectivamente. Para maiores detalhes sobre a maneira como essas instâncias foram geradas e suas propriedades, aconselhamos ao leitor interessado a leitura dos artigos originais onde as mesmas foram apresentadas.

Os testes foram processados utilizando 1000 iterações para cada instância e 10 execuções com sementes diferentes. As instâncias mais difíceis foram executadas até encontrar a solução ótima ou até atingir um máximo de 1 milhão de iterações. Macambira [4] implementou um algoritmo exato que forneceu as soluções ótimas para as duas categorias C1 e C2.

Nas tabelas 1 e 2, mostramos o resumo dos resultados obtidos para as categorias C1 e C2, respectivamente. Nessas tabelas, a primeira coluna mostra o tipo das instâncias de acordo com as suas características como explicado acima. A segunda coluna trás as restrições de demanda para cada anel. A terceira coluna trás o número de soluções viáveis para cada categoria e tipo. A quarta e a quinta colunas mostram o número absoluto e percentual de soluções viáveis encontrado pelo GRASP. As próximas duas colunas mostram esses números para as soluções ótimas encontradas. Finalmente, a última coluna mostra os tempos computacionais médios para cada classe de instâncias.

Na Tabela 1, podemos ver que o GRASP proposto encontrou soluções ótimas para todas as instâncias. Os tempos médios de execução foram menores do que 0,04 segundos.

**Tabela 1: Resultados computacionais para a classe C1.**

| Instâncias | | | Soluções Viáveis | | Soluções Ótimas | | |
|---|---|---|---|---|---|---|---|
| **Tipo** | **B (Mbs)** | **# SV** | **# SV - G** | **# SV - G (%)** | **# SO - G** | **# SO - G (%)** | **Tempo Médio (s)** |
| GL | 155 | 23 | 23 | 100,0 | 23 | 100,0 | 0,027 |
| GH | 622 | 27 | 27 | 100,0 | 27 | 100,0 | 0,021 |
| RL | 155 | 28 | 28 | 100,0 | 28 | 100,0 | 0,035 |
| RH | 622 | 33 | 33 | 100,0 | 33 | 100,0 | 0,019 |

Na Tabela 2, mostramos os resultados obtidos para a segunda categoria de instâncias C2. Fazemos a seguinte observação a respeito dessa categoria: o GRASP encontrou soluções ótimas em 227 dos 230 problemas testes e os tempos de execução médios para o GRASP foram menores do que 1,1 segundos.

**Tabela 2: Resultados computacionais para a classe C2.**

| Instâncias | | | Soluções Viáveis | | Soluções Ótimas | | |
|---|---|---|---|---|---|---|---|
| **Tipo** | **B (Mbs)** | **# SV** | **# SV - G** | **# SV - G (%)** | **# SO - G** | **# SO - G (%)** | **Tempo Médio (s)** |
| GL | 155 | 70 | 70 | 100,0 | 70 | 100,0 | 0,314 |
| GH | 622 | 70 | 70 | 100,0 | 68 | 97,1 | 0,382 |
| RL | 155 | 70 | 69 | 98,6 | 69 | 98,6 | 1,044 |
| RH | 622 | 20 | 20 | 100,0 | 20 | 100,0 | 0,747 |

A Tabela 3 lista as instâncias para as quais o GRASP não encontrou a solução ótima. Além do número de iterações e do tempo de CPU para encontrar a melhor solução, a tabela lista o tempo total para executar 1000 iterações, bem como o *gap* (100[$z_{GRASP} - z_{exato} / z_{GRASP}$) entre o ótimo e a solução obtida pelo GRASP. Deve ser notado que, exceto para a instância new.RL_50_8.5 onde o algoritmo falhou em encontrar solução viável, o *gap* é mínimo, ou seja, 16.67% o que corresponde a apenas um anel a mais do que o número de anéis da solução ótima. Além disso, estas soluções foram obtidas em no máximo 1,09 segundos.

**Tabela 3: Resultados dos testes para as instâncias onde o GRASP não encontrou a solução ótima.**

| Instâncias | $z_{exato}$ | $z_{GRASP}$ | gap (%) | It. Melhor | Tempo Melhor | Tempo Total |
|---|---|---|---|---|---|---|
| New.GH_50_3.4 | 5 | 6 | 16.67 | 160 | 0,175 | 1,094 |
| New.GH_50_9.4 | 5 | 6 | 16.67 | 31 | 0,033 | 1,078 |
| New.RL_50_8.5 | 5 | - | - | - | - | - |

## 4 Conclusões

Neste trabalho, estudamos o problema de atribuição de localidades a anéis SONET. Propomos um método heurístico para a solução do problema. O método, um GRASP, foi descrito em detalhes. A qualidade das soluções heurísticas pode ser medida pelo limite inferior produzido por um método exato.

Os resultados computacionais obtidos para as instâncias de teste ilustraram a completa eficiência da heurística GRASP proposta em encontrar soluções ótimas para o SRAP. Dessa forma, nosso algoritmo é competitivo com a melhor heurística encontrada na literatura em termos de qualidade da solução e em termos de tempo computacional. Somente para uma instância o GRASP falhou em encontrar soluções viáveis. Os problemas para os quais não foi possível encontrar a solução ótima, o desvio percentual entre a solução obtida e o valor do ótimo foi mínimo. Considerando todas as instâncias, a melhor solução foi encontrada num tempo médio de menos do que 0,5 s.

## Referências


[1] R. Aringhieri and M. Dell'Amico. Solutions for the SONET ring assignment with capacity constraints, in Proc. of the 4th Metaheuristic International Conference, Porto, Portugal, 2001.

[2] T. A. Feo and M. G. C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization, 6:109-133, 1995.

[3] O. Goldshmidt, A. Laugier and E. V. Olinick, SONET/SDH ring assignment problem with capacity constraits, Discrete Applied Mathematics, 129:99-128, 2003.

[4] E. M. Macambira, Modelos e algoritmos de programação inteira no projeto de redes de telecomunicações, Tese de doutorado, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, maio, 2003.