GRASP com Memória Adaptativa na solução de um Problema de Roteamento de Veículos com Múltiplas Origens

Tiago Araújo Neves
Instituto de Computação - Universidade Federal Fluminense
Niterói – Rio de Janeiro
tneves@ic.uff.br

Luiz Satoru Ochi
Instituto de Computação - Universidade Federal Fluminense
Niterói – Rio de Janeiro
satoru@ic.uff.br

RESUMO

Na literatura de metaheurísticas, o termo *memória* foi sempre usado de forma explícita somente para o algoritmo de Busca Tabu, mas outros métodos como Algoritmo Genético, *Scatter Search* e Colônia de Formigas utilizam mecanismos que podem ser vistos como tipos de memória. Neste trabalho apresentamos um método iterativo com memória adaptativa utilizando conceitos da heurística GRASP na solução de um problema de roteamento e *scheduling* de veículos com múltiplos depósitos. Resultados experimentais ilustram a eficiência da versão adaptativa quando comparado a um GRASP padrão.

Palavras-chave. Metaheurísticas. Memória Adaptativa. GRASP. Problema de Roteamento e *Scheduling* de Veículos

ABSTRACT

In the metaheuristic literature the term *memory* was used explicitly for Tabu Search algorithm only, but some other as Genetic Algorithm, Scatter Search and Ant Colony System use mechanisms that can be considered as memories. In this paper we propose an adaptive memory programming method using concepts of GRASP for solving a Vehicle Routing and Scheduling Problem with Multiple Depots. Experimental results illustrate the effectiveness of the adaptive GRASP over standard GRASP.

Keywords. Metaheuristics, Adaptive Memory. GRASP. Vehicle Routing and Scheduling Problem.

1. Introdução

A literatura apresenta vários problemas de otimização relacionados à área de petróleo. Um destes problemas, está associado à manutenção de poços petrolíferos.

Quando um poço terrestre de extração de petróleo tem sua produção comprometida por falha mecânica do equipamento de extração é necessário, que se faça uma manutenção para assegurar o funcionamento deste poço. Esta manutenção é feita por um equipamento denominado sonda, que é um equipamento com custo de aquisição muito elevado. A cada dia parado, um poço deixa de coletar uma certa quantidade de óleo. Sendo assim, a manutenção deste poço deve ser feita o mais breve possível a fim de assegurar que este poço volte a produzir.

Contudo manter uma sonda por poço é economicamente inviável. O que ocorre na prática é que, a manutenção de vários poços deve ser efetuada por um número reduzido de sondas fazendo com que, uma mesma sonda execute a manutenção em vários poços durante um dado horizonte de planejamento.

Determinar que poços serão atendidos por uma determinada sonda e em que ordem estes poços deverão ser atendidos é um problema de otimização combinatória na área de petróleo encontrado na região nordeste do Brasil, conhecido como *Problema de Roteamento e Scheduling de Sondas de Manutenção* (PRSSM). O problema consiste em gerar rotas otimizadas para uma frota de sondas de manutenção que atende a um conjunto de poços petrolíferos terrestres para um período de tempo determinado.

O objetivo deste problema é definir, para cada horizonte de planejamento de T unidades de tempo, o melhor escalonamento de poços para as sondas disponíveis; minimizando a perda total de produção de petróleo associada aos poços que estão aguardando atendimento. Uma das justificativas para resolver este problema de forma otimizada, é a sua grande importância econômica, pois ao minimizar a quantidade de óleo que deixa de ser coletada, através de um bom escalonamento, aumenta-se à produção de forma muito significativa.

O PRSSM ainda é pouco explorado na literatura. Propostas para a sua solução encontra-se, como por exemplo em: [1] onde os autores apresentam uma heurística baseada em Colônia de Formigas, [2] apresentam uma heuristica VNS para o problema enquanto [18] em sua dissertação de mestrado, apresenta um estudo sobre a heurística GRASP aplicado ao PRSSM. Outra proposta usando GRASP tradicional é apresentada em [4].

Em termos de literatura de Problemas de Roteamento de Veículos (PRV), o PRSSM pode ser classificado como um modelo de Problema de Roteamento de Veículos com Múltiplos Depósitos (PRVMD), problemas estes, classificados como da classe NP-Difícil. Devido a sua elevada complexidade computacional, o caminho natural para a solução deste problema, é através de técnicas aproximadas ou heurísticas.

Dentre as metaheurísticas existentes, o GRASP tem produzido bons resultados para diferentes modelos de PRV [6]. Contudo uma das limitações desta técnica, é o fato das suas iterações serem independentes, ou seja, nenhuma informação das iterações anteriores é levada em conta no processo de construção de uma nova solução.

Tendo em vista que as técnicas de memória adaptativa, mostradas em [16], podem melhorar de forma considerável a qualidade das soluções encontradas por metaheurísticas, seu uso deve ser feito sempre que possível, com o intuito de aprimorar os resultados obtidos.

Estas técnicas conseguem melhorar os resultados obtidos por metaheurísticas através do uso inteligente de estruturas de memória adaptativa.

Tais estruturas são atualizadas durante o processo de busca, armazenando informações relevantes para construção de soluções de melhor qualidade, ou mesmo informações sobre espaços de busca previamente explorados.

Desta forma, colocamos como objetivo deste trabalho, aplicar técnicas de memória adaptativa numa heurística GRASP, na resolução do PRSSM.

2. Descrição do problema

A retirada de petróleo e gás do subsolo, seja em terra ou mar, é uma atividade de elevado custo, pois requer mão de obra especializada, alta tecnologia e equipamentos sofisticados. E por se tratarem de recursos não renováveis, as jazidas necessitam serem exploradas por métodos comprovadamente eficientes que garantam o melhor aproveitamento das mesmas.

Como dito em [1], em muitos casos, a elevação dos fluidos é feita de forma artificial com o uso de equipamentos especiais devido ao fato de que os poços nem sempre possuem pressão suficiente para que os fluidos atinjam a superfície, por isso, serviços como os de limpeza e manutenção dos equipamentos de elevação de fluidos são essenciais para se evitar paradas no processo de extração. Esses serviços são realizados por equipamentos denominados sondas, que estão disponíveis em um número limitado e muito pequeno se comparado à quantidade de poços que demandam serviços, devido ao seu alto custo de operação. Por isso, a manutenção imediata nem sempre é possível, o que provoca a ocorrência de uma fila de poços a espera de atendimento.

Quando é detectada uma falha nos equipamentos de algum poço, para que este volte ao seu funcionamento o mais breve possível, é feito um pedido de conserto. Em seguida é feito um planejamento de atendimento para cada sonda disponível em um dado período de tempo pré-estabelecido, e então essas sondas são designadas para a correção dos poços que aguardam conserto na ordem estabelecida pelo seu escalonamento (veja mais detalhes em [1]).

Desta forma, podemos colocar como objetivo do Problema de Roteamento de Sondas de Manutenção (PRSSM), encontrar o melhor escalonamento para as sondas disponíveis, minimizando a perda total de produção de petróleo associada aos poços que estão aguardando por atendimento. Entende-se por perda de produção de petróleo de um poço como sendo a vazão por unidade de tempo desse poço multiplicado pela quantidade de unidades de tempo que esse poço teve sua produção interrompida. Além disso, para gerar uma boa solução do problema, deve-se fazer um escalonamento de sondas para o conjunto de poços, considerando informações como: vazão do poço, sua posição em relação aos demais poços e o prazo limite para o seu atendimento.

O PRSSM pode ser descrito na estrutura de um grafo completo não direcionado G = G(P,E) onde $P=\{p_1,p_2,...,p_n\}$ representa os poços (vértices) que estão a espera de manutenção $e=\{(p_i,p_j) : i\neq j \ / \ p_i,p_j \in P\}$ representa as arestas ligando os pares de poços de P. Considere ainda o conjunto $S=\{s_1,s_2,...,s_k\}$ o conjunto de sondas disponíveis para atender aos poços que aguardam a manutenção. Cada poço possui: uma vazão v_i, que representa a quantidade de óleo produzido por este poço por unidade de tempo, um tempo de serviço ts_i, que representa o tempo estimado para que uma sonda realize a manutenção neste poço, e uma folga f_i, que representa o tempo limite para que este poço seja atendido. Esta última restrição, surge do fato de que cada poço independente da sua vazão, deve ser atendido dentro de um limite de tempo pré-definido pela empresa. Cada poço só pode ser atendido uma única vez e por uma única sonda. O objetivo é minimizar a vazão perdida (que deixa de ser coletada) satisfazendo as restrições do problema.

3. Memória adaptativa

De uma forma geral, a técnica de memória adaptativa durante o processo de execução de um algoritmo iterativo consiste em armazenar informações relevantes de soluções e/ou parâmetros e/ou estratégias já analisadas em iterações anteriores e utilizar estas informações de maneira a tentar melhorar a estratégia de busca de novas soluções.

Existem diferentes propósitos ou metas estabelecidas com a inserção de procedimentos de memória adaptativa numa heurística iterativa. Pode-se buscar a calibração de parâmetros do algoritmo, como por exemplo o parâmetro α que mede a cardinalidade da lista restrita de candidatos na etapa de construção do GRASP; memória para evitar movimentos que não se mostraram eficazes em iterações anteriores ou para incentivar movimentos que se mostraram muito eficazes nas iterações anteriores; memória para construir ou atualizar um conjunto de soluções elite; memória para calibrar determinada combinação de módulos (exemplo: algoritmos de construção + busca local) numa dada instancia de um problema; memória para criar determinados padrões (por exemplo, segmentos de uma solução; fixação de variáveis de uma solução, etc) com o propósito de filtrar as direções de busca nas iterações remanescentes.

O conceito de memória adaptativa tem relação com algoritmos de aprendizado, e portanto possui normalmente um caráter dinâmico. Ou seja, as informações contidas na memória devem sempre que possível, serem atualizadas. Para isso a cada iteração busca-se considerar informações relevantes desta etapa.

Às vezes é conveniente considerar somente informações recentes (*memória de curto prazo*). Em outros casos, manter informação sobre todo o espaço já verificado é extremamente necessário (*memória de longo prazo*). Todas estas questões devem ser levadas em conta no momento de atualizar a estrutura de memória utilizada.

As técnicas de memória adaptativa normalmente possuem uma descrição muito sucinta, porém, seu uso ideal na prática não é tão trivial. Questões como que atributos das soluções deve-se armazenar, que técnicas utilizar para combiná-los, por quanto tempo um atributo deve ser levado em consideração ainda são temas polêmicos e normalmente tratadas caso a caso [7][8][10][16][17].

Existem duas grandes vertentes sobre uso de memória adaptativa em heurísticas: A proposta por Glover [7], e a proposta por Taillard [16]. A primeira é mais específica para Busca Tabu, a segunda é mais genérica, e pode ser aplicada a diversas metaheurísticas.

4. Propostas

Para avaliar a eficiência das técnicas de memória adaptativa foram construídos duas heurísticas, sendo um GRASP simples (GRASP Puro) e uma versão que faz uso de memória chamado de GRASPWAM (GRASP *With Adaptive Memory*). Ambos utilizam o mesmo método construtivo e a mesma busca local.

O método construtivo aqui utilizado é um dos propostos em [18], e que obteve o melhor desempenho para o PRSSM abordado neste trabalho nos testes realizados. Denotamos este construtivo de C1.

```
 0 Procedimento C1()
 1   Solução sol =  nova Solução;
 2   indexSonda = 0;
 3   Enqunato houver poços sem atendimento Faça
 4       Lista lis = construirLRC(α, sol, indexSonda);
 5       Poço p = sorteio(lis);
 6       inserirPoçoNoFinalDaRota(p, indexSonda, sol);
 7       indexSonda = (indexSonda +1)% numTotalDeSondas
 8   Fim Enquanto
 9   Retorne sol;
10Fim C1
```

Figura 1. Pseudo-código do construtivo C1

A figura 1 apresenta o pseudo-código do construtivo C1. A cada iteração ele acrescenta um poço a uma rota da solução "sol" (linha 6). Para escolher o poço a ser inserido a cada iteração de C1, o algoritmo faz um sorteio entre os candidatos de uma lista restrita de candidatos (LRC). Para construir a LRC, é utilizado o procedimento `construirLRC` (linha 4) descrito na figura 2

.O procedimento constrói uma lista contendo os α * `numPoçosNãoAtendidos` poços mais aptos, onde α é o coeficiente de gulosidade do método e `numPoçosNãoAtendidos` é o número de poços que ainda não foram atendidos.

```
 0 Procedimento construirLRC (Flutuante α , Solução sol , Inteiro
indexSonda)
 1  Lista lis =  nova Lista;
 2  Poço p1 = ultimoPoçoDaRota(Solução, indexSonda);
 3  Para cada Poço p2 ainda não inserido Faça
 4     fator de ordenação de p2 = vazão(p2)/distância(p1,p2)
 5  Fim Para
 6  ordene(listaPoçosNãoInseridos)
 7  tamahoLis = α *   numPoçosNãoAtendidos
 8  z=0
 9  Enquanto z < tamanhoLis
 10    inserir(listaPoçosNãoInseridos[z], lis )
 11    z = z+1
 12 Fim Enquanto
 13 retorne lis;
 14Fim construirLRC
```

Figura 2. Pseudo-código do algoritmo construirLRC

A busca local utilizada neste trabalho, é baseada na melhor busca local proposta em [18] (BL3). A BL3 é uma busca local de duas vizinhanças com busca exaustiva, a primeira delas permuta posições de poços pertencentes à mesma sonda, a segunda re-aloca poços entre sondas diferentes. A diferença entre a busca local utilizada neste trabalho, chamada de BL4, e a BL3, é o critério de parada. A BL3 pára se a segunda vizinhança não produzir melhoras. Na BL4 o algoritmo pára se e somente se nenhuma das duas estruturas de vizinhança produzir melhora além de conter elementos específicos para o PRSSM abordado neste trabalho. Testes preliminares realizados indicam que, em média, a BL4 produz resultados médios melhores, partindo da mesma solução inicial, em 40% dos casos, sendo que nos outros 60% os resultados encontrados são idênticos aos encontrados pela BL3.

```
0 Procedimento BL4(Solução sol)
1  val = 0;
2  Enquanto val for diferente custo(Sol) Faça
3     val = custo(Sol);
4     vizinhança1(Sol);
5     vizinhança2(Sol);
7  Fim Enquanto
8Fim BL4
```

Figura 3 - Pseudo-código da busca local BL4

Na figura 3, no procedimento vizinhança1 (linha 4) é feita uma busca exaustiva onde são testados as trocas de posição de cada poço com todos os demais poços pertencentes à mesma sonda.

Na *vizinhança2* (linha 5), para cada poço, é testada a realocação para todas as posições de todas as sondas que não sejam a sonda onde o poço originalmente se encontrava, tratando-se assim de uma busca exaustiva.

O GRASPWAM faz uso de uma série de técnicas bem conhecidas de memória adaptativa, que serão citadas e explicadas a seguir.

Auto configuração do parâmetro α (coeficiente de *gulosidade* do algoritmo construtivo do GRASP): um dos problemas fundamentais de sintonia fina do GRASP é a escolha do α [10][11]. É preciso encontrar um valor que produza boas soluções mas que também possibilite que o algoritmo não fique preso em ótimos locais. Para resolver este problema foi proposto o modelo de GRASP reativo, que configura automaticamente o valor deste parâmetro de acordo com a instância trabalhada, através de uma série de tentativas de construção com diferentes valores para α. Ao final das tentativas, escolhe-se o valor mais adequado, e, no restante do tempo (ou das iterações remanescentes), o GRASP executará com este valor agora fixado. Neste trabalho, o valor de α escolhido é aquele que gerou a melhor solução durante a faze de treinamento.

```
0 Procedimento GRASP()
1   Solução melhor = nova Solução;
2   Enqunato não atingir o critério de parada Faça
3       Solução aux = C1();
4       BL4(aux);
5       se custo(aux) < custo(melhor) então
6           melhor = aux;
7   Fim Enquanto
8   Retorne melhor
9Fim GRASP
```

Figura 4 - Pseudo-código do algoritmo GRASP

Conjunto Elite e Reconexão de Caminhos: A reconexão de caminhos (RC) é uma técnica proposta inicialmente para a Busca Tabu e *Scatter Search* por Glover e Laguna [7][8]. A RC utiliza duas soluções extremas de boa qualidade, uma denominada *base* e a outra *guia,* para tentar encontrar uma solução intermediária que seja melhor que as duas soluções da extremidade. A idéia é fazer com que passo a passo, a solução base fique mais parecida com a solução guia

O conjunto elite é usado como repositório de soluções base e/ou guia as quais a RC será aplicado. Neste trabalho a RC é utilizado de maneira hierárquica. Durante os 10% finais do tempo de execução total (critério de parada do GRASP), todas as soluções produzidas pelas iterações GRASP são submetidas à RC com cada um dos membros presentes no conjunto elite nos dois sentidos (da base para a guia e vice versa). Se durante esse procedimento o conjunto elite for modificado, ao final dele é feito a RC entre os membros do conjunto elite, também nos dois sentidos.

Diversificação Reativo+Conjunto Elite: Também foi feita uma estratégia simples de diversificação que é a variação do parâmetro α de forma associada com a atualização do conjunto elite durante a execução. Esta variação é feita da seguinte maneira: durante a busca, é feita uma medição de quanto tempo o conjunto elite ficou estático (sem modificação). Toda vez que este tempo atinge um valor t (parâmetro de entrada), o valor de α é aumentado em 0,1, tornando o método menos guloso. Quando o Conjunto elite é atualizado, α retorna ao valor que possuía antes da primeira modificação.

```
0 Procedimento GRASPWAM()
1   encontrarMelhorAlpha();
2   Enqunato não atingir o critério de parada Faça
3       Solução aux = C1();
4       BL4(aux);
5       tentarInserirNoConjuntoElite(aux);
6       Se tempoSemMelhora > t entao
7             aumetar o valor de α
8       Se condição do path relink foi atingida
9             executarPathRelink(aux)
10  Fim enquanto
11  Retorne melhor solução do conjunto elite
12Fim GRASPWAM
```

Figura 5 - Pseudo-código do GRASPWAM

A figura 4 ilustra o pseudo-código do algoritmo GRASP puro sem nenhum tipo de memória. Ele constrói uma nova solução utilizando o procedimento C1 e melhora a solução construída utilizando a busca local BL4 enquanto o critério de parada não for atingido. A melhor solução encontrada durante o procedimento é armazenada e retornada ao final do algoritmo.

A figura 5 mostra o pseudo-código do GRASP com as estratégias de memória descrito anteriormente: GRASPWAM. No inicio ele encontra o melhor α para o construtivo C1, em seguida, a cada iteração, é construída uma nova solução utilizando o construtivo C1 e esta solução é melhorada utilizando a busca local BL4, exatamente como no GRASP. Logo a seguir são avaliadas duas condições. A primeira avalia se o algoritmo está preso em um ótimo local, verificando à quanto tempo não há melhora no conjunto elite. Se este tempo atingir um limite t, passado como parâmetro, então o algoritmo aumenta o valor de α na tentativa de escapar do ótimo local. O valor de α retorna ao valor que era no início da busca sempre que uma inserção no conjunto elite é bem sucedida. Este passo é feito no procedimento `tentarInserrirNoConjutoElite`().

A segunda condição testada é a que define se o metodo `executaPathRelink`() deve ser executado. O método deve ser executado se 90% do tempo total de busca (critério de parada do GRASP) já transcorreu.

```
0 Procedimento pathRelink(Solução base, Solução guia)
1   Solução aux = copiar(base);
2   Solução melhor = copiar(guia)
3   calcularDiferenças(aux, guia) ;
4   Enquanto houver diferenças entre aux e guia Faça
5       aux = desfazerUmaDiferença(aux, guia);
6       Se custo(aux) < custo(melhor)
7             melhor = aux;
8   Fim Enquanto
9   Retorne melhor
10Fim pathRelink
```

Figura 6 - Pseudo-código do algoritmo Reconexão de Caminhos

A figura 6 mostra o pseudo-código do algoritmo que implementa a técnica de RC. A idéia é tornar a solução base gradativamente mais parecida com a solução guia.

```
0 Procedimento executaPathRelink(Solução sol)
1   Para cada solução s1 no conjunto elite;
2       Solução aux = pathRelink(sol, s1)
3       tentarInserirNoConjuntoElite(aux);
4       aux = pathRelink(s1,sol);
5       tentarInserirNoConjuntoElite(aux);
6   Fim Para
7   Se o conjunto elite foi modificado
8       conjuntoElitePathRelink();
9 Fim executaPathRelink
```

Figura 7 - Pseudo-código do método executaPathRelink

A figura 7 mostra o pseudo-codigo do método `executaPathRelink`. A função deste método é executar a técnica de Reconexão de caminhos entre uma solução, passada como parâmetro, e as soluções presentes no conjunto elite. A execução da técnica é feita nos dois sentidos, uma vez que os passos da busca são diferentes. Caso o conjunto elite seja modificado durante a execução deste método, o método `conjuntoElitePathRelink` é chamado.

```
0Procedimento conjuntoElitePathRelink(Solução sol)
1   Para cada solução s1 no conjunto elite Faça;
2       Para cada solução s2 != de s1 Faça
3           Solução aux = pathRelink(s2, s1);
4           tentarInserirNoConjuntoElite(aux);
5           aux = pathRelink(s1,s2);
6           tentarInserirNoConjuntoElite(aux);
7       Fim Para
8   Fim Para
9 Fim conjuntoElitePathRelink
```

Figura 8 - Pseudo-código do método conjuntoElitePathRelink

```
0Procedimento executaPathRelink(Solução sol)
1   Para cada solução s1 no conjunto elite;
2       Solução aux = pathRelink(sol, s1)
3       tentarInserirNoConjuntoElite(aux);
4       aux = pathRelink(s1,sol);
5       tentarInserirNoConjuntoElite(aux);
6   Fim Para
7   Se o conjunto elite foi modificado
8       conjuntoElitePathRelink();
9 Fim executaPathRelink
```

Figura 9 - Pseudo-código do método executaPathRelink

A figura 8 apresenta o pseudo-código do método `conjuntoElitePathRelink`. Este método executa o método `pathRelink` entre as soluções do conjunto elite.

5. Resultados Computacionais

Os testes computacionais foram feitos em um computador Pentium 4 com 2.6 GHz com 512 MB de memória principal rodando um sistema operacional Linux versão 2.6.8-1.521, tendo como critério de para 1h (uma hora) de tempo de cpu.

Todos os componentes e classes foram desenvolvidos na linguagem de programação c++ com o compilador G++ versão 4.0.2 e utilizando as opções de compilação "-o3" e "-mcpu = pentium4".

O parâmetro α para o GRASP foi fixado em 0.1 e para o GRASPWAM ele foi calibrado para cada instância como descrito anteriormente.

Devido à inexistência de instâncias com características específicas, como, por exemplo, limites de tempo individuais e variação do tempo de atendimento de cada poço, foi construído um gerador de instâncias para o PRSSM abordado neste trabalho. Foram geradas 3 instâncias com 50 poços, 3 com 100 poços e 3 com 500 poços.

A seguir são descritas as tabelas de configuração dos parâmetros de acordo com as instâncias e os resultados obtidos.

Instância	t (segundos)
50 Poços	100
100 Poços	100
500 Poços	300

Tabela 1- Configuração do parâmetro t1 de acordo com as instâncias

Instância	GRASP		GRASPWAM	
	Melhor	Média	Melhor	Média
50-1	**42.682**	42.682	**42.682**	42.682
50-2	**45.516**	45.516	**45.516**	45.516
50-3	**41.623**	41.623	**41.623**	41.623
100-1	136.215	136.518	**135.074**	135.722
100-2	112.593	112.889	**111.974**	112.053
100-3	125.016	126.063	**124.515**	125.330
500-1	1.902.011	1.902.901	**1.876.079**	1.903.503
500-2	1.862.974	1.868.859	**1.861.564**	1.883.574
500-3	**1.863.145**	1.872.988	1.894.485	1.898.817
500-3*	1.855.022	1.863.925	**1.841.052**	1.857.685

Tabela 2 - Qualidade das soluções encontradas

A tabela 2 mostra os resultados computacionais médios em relação à qualidade das soluções encontradas pelos dois métodos (já que o critério de parado usado, foi um tempo limite). A coluna "*Melhor*" mostra o melhor elemento encontrado por cada metaheurística. A coluna "*Média*" mostra o resultado médio de 3 execuções para cada instância. Pelos resultados ilustrados na tabela 2, verifica-se uma nítida superioridade da versão adaptativa sobre a versão pura do GRASP. A única instância onde a versão adaptativa a princípio perde, é numa das instâncias com 500 vértices (500-3). Neste caso, observamos que o tempo limite adotado foi muito pequeno, não permitindo que a versão GRASPWAM utilizasse forma adequada todos os seus módulos de treinamento, bem como a Reconexão de Caminhos. Em testes suplementares dobrando o tempo de execução permitida observa-se também neste caso, a superioridade da versão adaptativa. A linha 500-3* mostra os resultados obtidos com o tempo de execução igual a 2h (duas horas) para ambas as metaheurísticas.

6. Conclusões e Trabalhos Futuros

Este trabalho colocou como objetivo principal, desenvolver uma análise experimental empírica comparando o desempenho de um GRASP puro com o um GRASP usando diferentes tipos de memória adaptativa (GRASPWAM).

Para isso, partimos da versão GRASP puro proposto em [18] que obteve o melhor desempenho para o PRSSM abordado neste trabalho, e a esta versão, acrescentamos diferentes tipos de memória tais como: parâmetro α variável (GRASP Reativo); módulo de reconexão de caminhos (RC) *(path relinking)*; e um terceiro procedimento conjugando conceitos de GRASP Reativo com a atualização do conjunto elite da RC.

Nos testes realizados, observamos que a versão adaptativa obtém em quase todos os testes, uma solução de melhor qualidade num dado tempo de execução previamente definido. Somente numa das instâncias (*500-3*) verificamos que a versão adaptativa as vezes pode exigir um tempo um pouco maior para justificar o uso de todos os seus módulos.

De qualquer forma, podemos mesmo de forma empírica concluir que dando um tempo suficiente, a versão adaptativa tende sempre a produzir soluções de melhor qualidade do que as versões tradicionais do GRASP.

Como trabalhos futuros podemos sugerir o uso de outras técnicas de memória adaptativa tais como perturbações no algoritmo construtivo [6], construção de vocabulário [7], e mineração de dados [12]. Outra sugestão já em andamento, é avaliar os algoritmos aqui propostos para outras instâncias e o uso de outros critérios de parada para o GRASP, tais como um valor alvo e número Maximo de iterações permitidas.

7. Referências

[1] Aloise, D., Noronha, T.F., Maia, R. S., Bittencourt, V.G., & Aloise, D.J. (2002), Heurística de colônia de formigas com *path-relinking* para o problema de otimização da alocação de sondas de produção terrestre. *Anais do XXXIV Simpósio Brasileiro de Pesquisa Operacional* (SBPO), Rio de Janeiro.

[2] Aloise , D., Aloise, D. J., Rocha, C. T. M., Ribeiro Filho, J. C., Moura, L. S. S. & Ribeiro, C. C. (2006), Scheduling Workover Rigs for Onshore Oil Production. *Discrete Applied Mathematics*, 154, 695-702.

[3] Aiex, R. M., Resende, M. G. C. & Ribeiro, C. C. (2002), Probability distribution of solution time in GRASP: an experimental investigation. *Journal of Heuristics*, 8, 343-373.

[4] Costa, L. R. (2005), Soluções para o Problema de Otimização de Itinerário de Sondas, *Dissertação de Mestrado em Engenharia de Produção,* Universidade Federal do Rio de Janeiro UFRJ.

[5] Feo, T. & Resende, M. G. C. (1995), Greedy Randomized Adaptive Search Procedure. *Journal of Global Optimization,* 6: 109-133.

[6] Festa, P. & Resende, M. G. C. (2002), GRASP: An annotated bibliography, In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, 325-367. Kluwer.

[7] Glover, F. & Laguna, M. (1998), Tabu Search. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[8] Glover, F., Laguna, M. & Marti, R. (2000), Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39, 653-684.

[9] Hansen, P., & Mladenovic, N. (2005), Variable Neighborhood Search. To appear in *State of the Art Handbook of Metaheuristics*. Kluwer

[10] Prais, M. & Ribeiro, C. C. (2000), Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing,* 12, 164-176.

[11] Resende, M. & Ribeiro, C. C. (2002), Greedy randomized adaptive search procedures (GRASP). In: *Handbook of Metaheuristics* [edited by F. Glover and G. Kochenberger], Kluwer Academic Publishers, 219-249.

[12] Santos, H. G., Marinho, E. H., Ochi, L. S., & Drummond, L. M. A. (2006), Combining an Evolutionary Algorithm with Data Mining to solve a Vehicle Routing Problem. Status: To appear in *Neurocomputing Journal* - ELSEVIER, 2006.

[13] Silva, G. C., Andrade, M. Q., Ochi, L. S., Martins, S. L., & Plastino, A. (2006), New heuristics for the Maximum Diversity Problem. To appear in *Journal of Heuristics* – SPRINGER.

[14] Silva, G. C., Ochi, L. S., & Martins, S. L. (2006), Proposta e Avaliação de heurísticas GRASP para o Problema da Diversidade Máxima. Status: aceito para publicação integral na Revista *Pesquisa Operacional* em 11/2005.

[15] Souza, M. J. F., Maculan, N. & Ochi, L. S. (2003), A GRASP-Tabu Search algorithm for solving School Timetabling Problems, In Combinatorial Optimization Book Series, Metaheuristics: Computer Decision-Making, vol. 15**,** chapter 31, 659-672, D. Z. Du and P. M. Pardalos (serie editors), Kluwer.

[16] Taillard, E. , Gambardella, L., Geandreau, M., Potvin, J.-Y. (2001), Adaptive Memory Programing: An Unified View of Metaheuristics. *European Journal of Operational Research*, 135, 1-16.

[17] Talbi, E.G. (2002), A Taxinomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8, 541-564.

[18] Trindade, V. A. (2005), Desenvolvimento e Análise Experimental da Metaheurística GRASP para um Problema de Planejamento de Sondas de Manutenção, *Dissertação de Mestrado em Ciência da Computação,* Universidade Federal Fluminense UFF.

[19] Whithey, D. (2001), An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43, 817-831.