Desenvolvimento e Análise Experimental da Metaheurística GRASP para um Problema de Planejamento de Sondas de Manutenção

Viviane de Aragão Trindade

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Otimização e Inteligência Artificial.

Orientador: Luiz Satoru Ochi

Niterói, Outubro de 2005.

Viviane de Aragão Trindade

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Otimização e Inteligência Artificial.

Aprovada por:

Prof. Luiz Satoru Ochi / IC-UFF (Presidente)

Profa. Lilian Markenzon / UFRJ

Profa. Luciana Salete Buriol / UFSM

Profa. Marley Maria Bernardes Rebuzzi Vellasco / PUC - Rio

Prof. Virgílio José Martins Ferreira Filho / UFRJ

Niterói, Outubro de 2005.

A meus pais, minha irmã e minha avó,

com amor.

Agradecimentos

Agradeço a meus pais, Amélia e Júlio, minha irmã, Marina, e minha avó, Madalena, por todo o carinho, amor, apoio e encorajamento que sempre deram na minha vida, a vocês todo e qualquer agradecimento será pouco.

Ao meu orientador, Luiz Satoru Ochi, por acreditar em mim, desde a iniciação científica, pela amizade, dedicação, carinho e crescimento durante todos esses anos.

Aos professores que fizeram parte da banca examinadora, pelas valiosas sugestões e contribuições a essa tese.

Aos amigos que fiz durante o mestrado na UFF, em especial aos amigos Marcos, Haroldo, Adria, Ivairton, Jacques, Lucas, Luis, Leandro e Stenio, pelas ajudas e conselhos.

Aos meus amigos de todas as horas que estão sempre presente na minha vida e no meu coração, em especial Adriana, Marcelo e Fernando Silveira, pela amizade incondicional e pela constante torcida.

A todos os funcionários do Instituto de Computação, em especial à Isabela, à Angela e ao Carlinhos.

Ao Instituto de Computação desta Universidade pela oportunidade de desenvolver meus estudos.

À Coordenação de Aperfeiçamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro.

Resumo da Tese apresentada à UFF como requisito parcial para a obtenção do grau de Mestre em Computação (M.Sc.)

Desenvolvimento e Análise Experimental da Metaheurística GRASP para um Problema de Planejamento de Sondas de Manutenção

Viviane de Aragão Trindade
Outubro/2005

Orientador: Luiz Satoru Ochi
Programa de Pós-Graduação em Computação

Nos campos petrolíferos terrestre existem dois tipos de poços: os surgentes e os não surgentes. O primeiro possui pressão suficiente para que seu petróleo atinja a superfície; o segundo, foco desse trabalho, não possui, e por isso necessita de equipamentos especiais para realizar a extração. Esses equipamentos necessitam constantemente de serviços, tais como: manutenção, limpeza e conserto, que são prestados por sondas de manutenção, disponíveis em um número limitado. Este trabalho aborda o problema de roteamento de sondas de manutenção (PRSM), que tem como objetivo encontrar o melhor escalonamento para cada sonda disponível, minimizando a vazão perdida associada com os poços que estão esperando por atendimento. Este é um problema real encontrado na região nordeste do Brasil e pode ser visto como uma nova variante dos problemas de roteamento de uma frota de veículos com múltiplos depósitos (PRVMD). Para a solução deste problema, são propostas: uma formulação matemática, descrevendo o PRSM como um problema de programação linerar inteira, e heurísticas utilizando o conceito da metaheurística GRASP, incluindo diferentes tipos de versões adaptativas, módulos de Reconexão de Caminhos e Filtro.

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Development and Experimental Analysis of GRASP Metaheuristics for a Workover Rigs Scheduling Problem

Viviane de Aragão Trindade

October/2005

Advisors: Luiz Satoru Ochi
Department: Computer Science

On the onshore fields there are two kinds of wells: those that have enough pressure to make its oil reach the surface and those that don't, and so they need special equipment to perform artificial lift methods. The second kind is the focus of this work. Those equipments need maintenance services such as cleaning, reinstatement, stimulation and others that are performed by workover rigs, avaliable on a limited number. This work approaches the scheduling workover rigs problem (SWRP) for onshore oil production, that has as its objective find the best schedule for the available workover rigs, minimizing the production loss associated with the wells that are waiting for service. This is a real problem that can be found on northeast area of Brazil and it can be seen as a variant of multi depot vehicle routing problems (MDVRP). To solve this problem it is proposed: a mathematic formulation, describing the SWRP as a linear programming problem, in addition to GRASP heuristics, including adaptative versions, Path Relinking and filter mothods.

Palavras-chave

1. Problema de Roteamento de Sondas de Manutenção
2. Metaheurística GRASP
3. Heurísticas de construção e busca local

Glossário

GRASP : *Greedy Randomized Adaptive Search Procedure*;

PRSM : Problema de Roteamento de Sondas de Manutenção;

LRC : Lista Restrita de Candidatos;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este trabalho apresenta propostas para a solução de um problema de otimização na área de petróleo encontrado na região nordeste do Brasil, conhecido como *Problema de Roteamento de Sondas de Manutenção* (PRSM). 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 abordar este problema, se refere a sua grande importância econômica, pois , pode-se notar ao longo deste trabalho, que se ao invés de quantidade de petróleo deixado de ser coletado num período, pensarmos em termos de valores financeiros envolvidos, concluímos que a economia resultante de um bom escalonamento de sondas é muito significativa.

Contudo, o PRSM ainda é pouco explorado na literatura. Este problema possui , entretanto, alguma similaridade com alguns modelos clássicos de roteamento de veículos da literatura, tal como o Problema de Roteamento com Múltiplos Depósitos (PRMD). Ainda assim, as diferenças encontradas entre estes modelos (PRSM e PRMD) são significativas e fazem com que uma simples adaptação dos métodos existentes para o PRMD não seja justificável na prática para serem aplicados ao PRSM.

Desta forma, apresentamos neste trabalho diferentes propostas inovadoras para a so-

lução do PRSM, contribuições estas, compostas de métodos eficazes de construção e de busca local para serem incorporadas a metaheurística GRASP.

São propostas desde versões clássicas da heuristica GRASP, com iterações independentes , até versões mais sofisticadas incorporando conceitos de memória nesta metaheuristica. Neste caso, são geradas formas híbridas do GRASP procurando tornar este método adaptativo através do uso de algum tipo de informações coletadas de iterações anteriores.

Neste contexto, são propostas versões GRASP incorporando módulos de Reconexão por Caminhos (*Path Relinking*), em que informações relevantes obtidas em iterações passadas são armazenadas na forma de um conjunto de soluções elite. Adicionalmente outras versões adaptativas são propostas, em que a idéia é, neste caso, efetuar uma fase de treinamento com diferentes tipos de: construção, busca local e calibração de parâmetros e, posteriormente, fixar as melhores combinações para as iterações restantes do GRASP. É também proposta a inclusão de um módulo de filtro na fase de construção do GRASP.

Extensos resultados computacionais com diferentes tipos de instâncias e análises mostram o potencial das versões híbridas propostas, com destaque para as versões reunindo conceitos de: reconexão por caminhos, memória adaptativa e filtro.

Adicionalmente, para avaliar as soluções heurísticas em pequenas instâncias, soluções ótimas são geradas por métodos exatos , utilizando uma formulação matemática do PRSM (outra proposta desse trabalho), descrevendo-o como um problema de programação linear inteira.

Este trabalho está dividido da seguinte forma: no capítulo 2 é apresentada tanto a definição do Problema de Roteamento de Sondas de Manutenção, quanto algumas propostas para este problema e para problemas similares existentes na literatura.

No capítulo 3, são apresentados os conceitos sobre a metaheurística GRASP e algumas técnicas híbridas utilizadas para tentar melhorar os resultados obtidos por esta metaheurística.

No capítulo 4, são detalhadas as propostas desse trabalho, tais como : algoritmos e a formulação matemática. Já no capítulo 5 , os resultados e análise dos mesmos.

Por fim, no capítulo 6 , são apresentadas as conclusões resultantes deste trabalho bem

como propostas de trabalhos futuros.

Capítulo 2

Problema de Roteamento de Sondas de Manutenção (PRSM)

2.1 Descrição do Problema

O petróleo bruto, que é considerado uma fonte de energia não renovável de origem fóssil, possui em sua composição uma cadeia de hidrocarbonetos cujas frações leves formam os gases e as frações pesadas formam o óleo cru. O que define os diversos tipos de petróleo existentes é a distribuição destes percentuais de hidrocarbonetos.

A retirada de hidrocarbonetos (óleo e gás) do subsolo, seja em terra (*onshore*) ou mar (*offshore*), é uma atividade de elevado custo, pois requer mão de obra especializada, alta tecnologia e equipamentos sofisticados. E por se tratar de recursos não renováveis, as jazidas necessitam ser exploradas por métodos comprovadamente eficientes que garantam o melhor aproveitamento dos mesmos.

A bacia Potiguar terrestre de petróleo e gás, que se localiza entre o Rio Grande do Norte e o Ceará (RN-CE), é responsável por cerca de 10% da produção nacional [1]. Esta bacia possui 77 campos petrolíferos (com número variável de poços) e abrange aproximadamente 4.500 poços com produção em torno de 18.300 m^3/dia, o que faz esta bacia estar em primeiro lugar em termos de produção terrestre no Brasil [segundo dados da PETROBRAS, RN/CE] [2]. A elevação dos fluidos, na maioria desses poços, é feita de forma artificial com o uso de equipamentos especiais, pois cerca de 98% dos poços petrolíferos dessa bacia são não surgentes, ou seja, não possuem pressão suficiente para que os fluidos atinjam a superfície [1].

Serviços como os de limpeza e manutenção dos equipamentos de elevação de fluidos são essenciais para evitar paradas na extração dos hidrocarbonetos. Esses serviços são realizados por uma equipe de manutenção, denominada *Sondas de Produção Terrestre - SPT* [1, 2], 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 [3].

Quando é detectada uma falha nos equipamentos de algum poço, para que este volte ao seu funcionamento o mais breve possível, um pedido de conserto é enviado à empresa responsável por gerenciar essas sondas. É feito então 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.

Diante desses fatos percebe-se a importância do planejamento otimizado da manutenção desses poços, já que uma vez mal planejada, atrasará o cronograma de produção de toda a reserva e afetará economicamente de maneira significativa a relação custo/produção da mesma.

Cada sonda de manutenção tem a capacidade de realizar um conjunto de serviços. Quando a empresa responsável por gerenciar essas sondas recebe pedidos de serviços para determinados poços, esta deve ter a preocupação de alocar uma sonda s_k apta para atender a um determinado poço p_i.

O escalonamento dessas sondas é feito para um período de atendimento de T unidades de tempo, no entanto, embora a princípio a empresa possa estimar o tempo para a realização de determinado serviço, este tempo pode na prática se tornar bem maior. Portanto, alguns poços podem não ser atendidos no atual período de planejamento e estes devem então entrar no próximo período de planejamento.

Existe, porém, uma restrição de que cada serviço (poço) deve ser necessariamente atendido em um tempo máximo de dias ($PRAZO$) previamente estipulado.

Observamos ainda que se um poço p_i possui uma vazão grande v_i e fica esperando por atendimento por muito tempo, essa vazão v_i está deixando de ser coletada, e consequente-

mente, está sendo perdida. Por outro lado, se uma sonda s_j vai atender a um poço p_i com uma elevada vazão v_i, mas que se encontra muito distante da posição atual da sonda s_j, enquanto s_j percorre o caminho até p_i, s_j pode estar deixando de atender muitos poços com vazão menor, porém mais próximos a s_j e com isso demandariam menos tempo para serem atendidos.

Observamos então que, para gerar uma boa solução do problema, deve-se considerar um escalonamento de sondas para o conjunto de poços, considerando informações como: vazão do poço, tipo de serviço requisitado, sua posição em relação aos demais poços e o prazo limite existente para o seu atendimento.

Desta forma, podemos colocar como objetivo do Problema de Roteamento de Sondas de Manutenção (PRSM) encontrar, para cada horizonte de planejamento de T unidades de tempo, 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 [4].

Por se tratar de um problema de otimização combinatória de difícil resolução, é inviável abordar o PRSM de forma exata, visto que para tal problema com as dimensões reais o universo de soluções é muito extenso. Ou seja, efetuar uma busca exaustiva neste grande universo de soluções se torna inviável em termos de tempo de processamento requerido, pois os resultados computacionais obtidos são utilizados para auxiliar a tomada de decisão em processos operacionais em tempo real [2].

O PRSM pode ser definido da seguinte forma: Dado um grafo completo não direcionado $G = (P, E)$, onde $P = \{p_1, p_2, ..., p_n\}$ representa um conjunto de poços (vértices) que estão a espera de algum serviço e $E = \{(p_i, p_j) : i \neq j / p_i, p_j \in P\}$ representa as arestas ligando cada dois poços de P. Cada vértice p_i possui associado uma vazão $v(i)$, que representa a perda de vazão deste poço por unidade de tempo, e um tempo de serviço $ts(i)$, que representa o tempo estimado que uma sonda s_w leva para executar o serviço requisitado pelo poço p_i. Considere também $S = \{s_1, s_2, ..., s_k\}$ como sendo o conjunto de sondas disponíveis para atender os poços que aguardam por atendimento e $P_w \subset P$ o conjunto de poços que podem ser atendidos pela sonda s_w. Cada poço deve ser atendido

uma única vez e por uma única sonda. O objetivo é minimizar a vazão de óleo perdida (que deixou de ser coletada) satisfazendo as restrições do problema.

Exemplo

Um exemplo simples é mostrado nas figuras 2.1, 2.2 e 2.3 para a melhor compreensão do problema. A simulação do planejamento do percurso de duas sondas é feita com o intuito de deixar claro os cálculos envolvidos no problema, como da vazão perdida e tempo gasto por cada sonda para realizar o atendimento de todo o período de planejamento. Lembrando ainda que nenhum algoritmo de escolha de poços será considerado neste exemplo, apenas vamos selecionar os poços ao acaso para exemplificar os cálculos.

Considerando a seguinte situação hipotética: sete poços, dispostos como mostra a figura 2.1, solicitaram atendimento para um período de planejamento de 30 unidades de tempo e 2 sondas estarão disponíveis para atender estes poços.

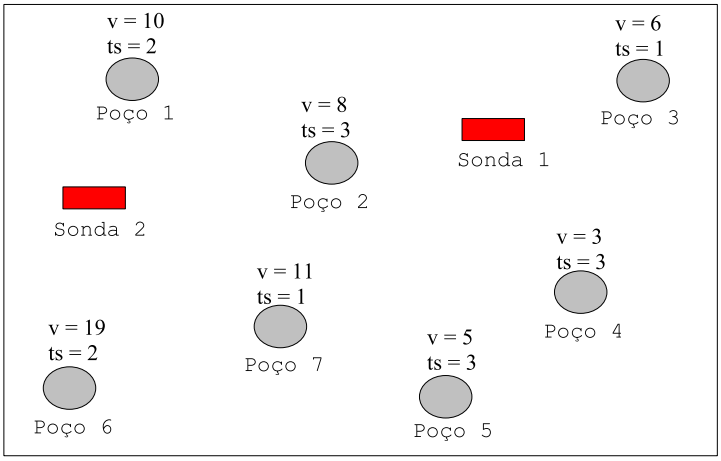

Figura 2.1: Localização dos poços a serem atendidos e das duas sondas disponíveis

O grafo associado é completo, ou seja, cada poço possui um caminho direto para todos os outros poços. Cada sonda está localizada no início de cada período de planejamento no último poço atendido no período anterior. Para cada poço da figura 2.1 são apresentados os respectivos valores de vazão v e tempo de serviço ts.

Assim, se a *sonda* s_1 tem como primeiro poço a atender o *poço* p_2 e a *sonda* s_2 o *poço* p_1, conforme nos mostra a figura 2.2, considerando o tempo de percurso t_{ij} entre cada sonda s_i, no início no período de planejamento, e o primeiro poço p_j de cada uma delas, mostrado na figura em unidades de tempo, estas terão como vazão perdida o seguinte

cálculo:

Temos como valor inicial para os cálculos :

- $tempo\_gasto(s_1) = 0$ e $tempo\_gasto(s_2) = 0$
- vazão$_perdida(s_1) = 0$ e vazão$_perdida(s_2) = 0$

sonda 1:

- $tempo\_gasto(s_1) = tempo\_gasto(s_1) + t(s_1, p_2) + ts(p_2) = 0 + 3 + 3 = 6$
- vazão$_perdida(s_1) =$ vazão$_perdida(s_1) + tempo_gasto(s_1) * v(p_2) = 0 + 6 * 8 = 48$

sonda 2:

- $tempo\_gasto(s_2) = tempo\_gasto(s_2) + t(s_2, p_1) + ts(p_1) = 0 + 1 + 2 = 3$
- vazão$_perdida(s_2) =$ vazão$_perdida(s_2) + tempo_gasto(s_2) * v(p_1) = 0 + 3 * 10 = 30$

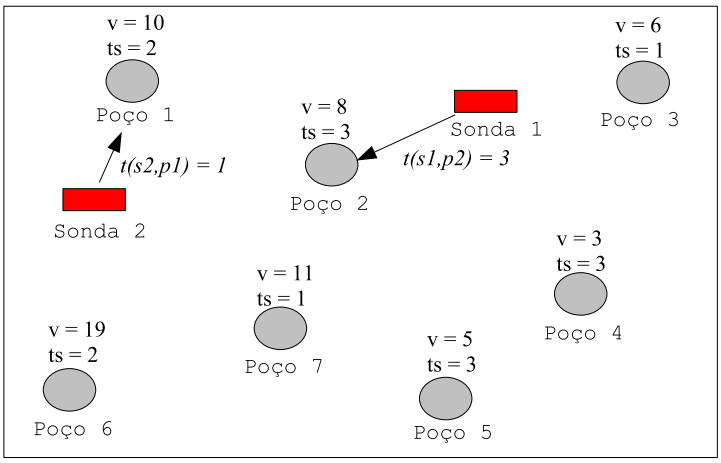

Figura 2.2: Alocação inicial de cada sonda

Depois de atender o *poço* p_2, suponha que a *sonda* s_1 atende ao *poço* p_6 conforme a figura 2.3. O cálculo da vazão perdida e do tempo gasto para esta sonda é então dada da seguinte maneira:

sonda 1:

- $tempo\_gasto(s_1) = tempo\_gasto(s_1) + t(p_2, p_6) + ts(p_6) = 6 + 7 + 2 = 15$
- vazão$_perdida(s_1) =$ vazão$_perdida(s_1) + tempo_gasto(s_1) * v(p_6) = 48 + 15 * 19 = 333$

sonda 2:

- $tempo\_gasto(s_2) = tempo\_gasto(s_2) + t(p_1, p_3) + ts(p_3) = 3 + 11 + 1 = 15$
- vazão$_perdida(s_2) =$ vazão$_perdida(s_2) + tempo_gasto(s_2) * v(p_1) = 30 + 15 * 6 = 120$

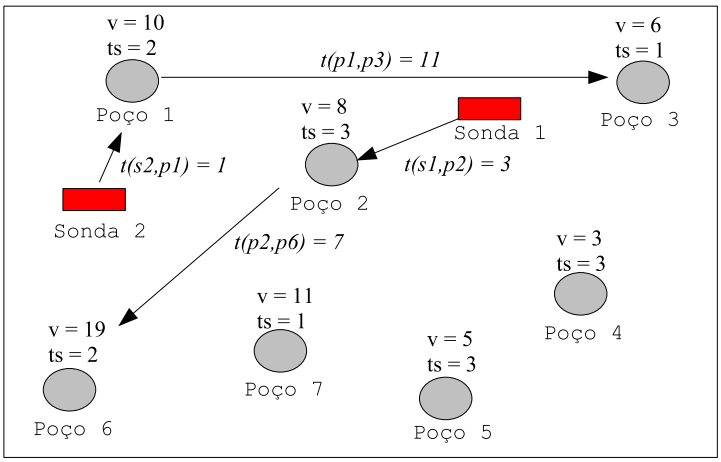

Figura 2.3: Alocações iniciais de cada sonda

Na resolução do PRSM este procedimento é repetido sucessivamente com base em alguma política de escolha do próximo poço a ser atendido por cada sonda, até que todos os poços que requisitaram atendimento tenham sido alocados. O objetivo do problema é minimizar a vazão total perdida, ou seja, a soma de vazões perdidas por todas as sondas envolvidas no horizonte de planejamento.

2.2 Literatura Existente

O Problema de Roteamento de Veículos (PRV) é bastante explorado na literatura. O estado da arte deste problema pode ser visto em Renaud *et al.* [5], Tillman [6], Wren e Holliday [7], Desrochers *et al.* [8], Salhi e Nagy [9], Min *et al.* [10], Laporte *et al.* [11]

entre outros. Tanto Laporte *et al.* [11] quanto Desrochers *et al.* [8] abordam o PRV básico. Já os demais trabalhos abordam o PRV de múltiplos depósitos (PRVMD).

O PRV pode ser representado na estrutura de um grafo $G = (V, E)$, onde V é um conjunto de vértices, $V = \{0, 1, ..., n\}$, e $E = \{(i, j) : i, j \in V\}$ um conjunto de arestas que ligam dois vértices de V. O vértice $i = 0$ representa o depósito (origem) e os demais vértices, os clientes. Associada a cada vértice cliente existe uma demanda e a cada aresta uma distância, ambas não negativas. As demandas são atendidas por uma frota de m veículos, localizados na origem, e cada cliente é visitado apenas uma única vez por um único veículo, onde a sua demanda deve ser atendida. O PRV consiste em definir m rotas (uma para cada veículo), saindo do depósito e voltando para este ao final de cada rota, tendo como objetivo minimizar a distância total percorrida pelos m veículos e atendendo às seguintes restrições: a demanda da rota não deve ultrapassar a capacidade de cada veículo e a distância total de cada rota não deve ultrapassar uma distância máxima pré estabelecida. Em alguns problemas esse número m de veículos é pré estabelecido e em outros é uma variável do problema [8, 9, 10, 11].

No PRV de múltiplos depósitos (PRVMD), considera-se o caso em que existem $k > 1$ depósitos, e cada veículo deve iniciar e finalizar sua rota no mesmo depósito [12].

O PRSM abordado nesta dissertação é um dos modelos de Problemas de Roteamento de uma frota de veículos com múltiplos depósitos (origens) (PRVMD). No entanto, todos os modelos de PRVMD existentes na literatura diferem substancialmente do PRSM. A similaridade entre os mesmos é que ambos possuem múltiplas origens e geram um conjunto de rotas. Por outro lado, as diferenças entre o PRSM e os demais PRVMD, são tais que:

- O PRSM não gera rotas fechadas, ou seja, as sondas não retornam ao seu poço origem ao final do seu escalonamento, ao contrário da maioria dos PRVMD.
- O PRSM, ao contrário da maioria dos modelos de PRVMD, possui restrições temporais (*scheduling*), ou seja, existem tempos associados pré-definidos e tempos a se determinar (variáveis).

A literatura apresentada para o PRVMD possui um número significativo de contribuições [5, 6, 7, 9, 10], mas poucas contribuições existem para o PRSM [3, 4, 13].

Tanto o PRVMD como o PRSM são variantes do Problema de Roteamento de Veículos tradicional (PRV) que é classificado como NP-difícil [14].

Apesar de não existir na literatura muitos trabalhos sobre o PRSM, algumas heurísticas já foram propostas para resolvê-lo, como uma heurística usando conceitos de Colônia de Formigas (*Ant Colony System*) [4] e um VNS (*Variable Neighborhood Search*) [3]. Nestes trabalhos, os autores usaram conjuntos de instâncias baseadas em características do modelo real brasileiro. No entanto estas instâncias não estão disponíveis em bibliotecas públicas, dificultando possíveis comparações com outras propostas.

Duas heurísticas de colônia de formigas foram propostas em [4]. Primeiro, foram elaborados dois algoritmos de construção: um simples, no qual vai se passando de sonda por sonda e cada uma faz a escolha de um poço para ser colocado em sua lista de atendimento, baseando-se em um fator de prioridade, até que não haja mais poços sem atendimento; o outro algoritmo difere do primeiro pelo fato de que quando um poço é adicionado a uma rota, este não é imediatamente retirado da lista de poços disponíveis. Neste, o poço continua na lista, podendo ser escolhido por outras sondas e permanecendo apenas na lista da sonda que resultar em um menor tempo de atendimento para o mesmo. O poço só é retirado da lista de poços disponíveis quando nenhuma outra sonda consegue atender ao poço com um tempo inferior à sonda para a qual ele está escalonado. Ambas heurísticas utilizam uma busca local para refinar a solução, esta busca faz trocas a cada par de poços de uma mesma sonda, quando encontra melhora, a solução melhorada passa a ser a solução corrente. As trocas continuam até que não haja mais melhora.

Na heurística VNS proposta em [3] uma solução inicial é gerada com um algoritmo construtivo onde para cada sonda, até que não haja mais poços esperando por alocação, seleciona-se um poço que tenha menor vazão perdida se alocado para a sonda em questão. Depois é aplicada nessa solução uma análise em nove vizinhanças usando a estrutura VNS [15, 16, 17, 18]. As vizinhanças são as seguintes e executadas nesta ordem: V1)Duas sondas trocam suas rotas; ou seja, a sonda s_k passa a percorrer a rota da sonda s_w e a sonda s_w da sonda s_k, V2)Troca de posição dois poços que estão alocados em uma mesma sonda, V3)Troca de posição dois poços que estão em sondas diferentes, V4)Um poço que está alocado para uma sonda é alocado para qualquer posição de alguma outra sonda, V5)Aplica dois movimentos consecutivos da vizinhança V2, V6)Aplica dois movimentos

consecutivos da vizinhança V3, V7)Aplica três movimentos consecutivos da vizinhança V3, V8)Aplica dois movimentos consecutivos da vizinhança V4 e V9)Aplica três movimentos consecutivos da vizinhança V4.

Os dois artigos da literatura [3] e [4] não podem ser comparados, pois estes não utilizam as mesmas instâncias para apresentar os resultados conseguidos pelos algoritmos propostos nestes. Em [3] os autores apresentam uma heurística usando conceitos da metaheurística VNS, enquanto heurísticas usando conceitos do método Colônia de Formigas são apresentadas por [4].

Em relação ao modelo PRV básico como para o PRVMD, como já citado anteriormente, existem muitas constribuições na literatura. Portanto, iremos a seguir expor de forma resumida algumas das contribuições mais recentes. Primeiramente estão expostos os artigos de PRV e posteriormente os de PRVMD.

Em Laporte *et al.* [11], é feita uma revisão das famílias de heurísticas mais importantes para a resolução do PRV, englobando heurísticas clássicas e a Busca Tabu, já que considerando as metaheurísticas, a Busca Tabu é a que tem apresentado os melhores resultados para o problema. Das heurísticas clássicas, algumas que utilizam critérios de economia para fundir rotas existentes são abordadas, tanto em versão paralela quanto em versão sequencial. Outras também abordadas, são as que atribuem gradualmente vértices a cada rota de um veículo utilizando um critério de custo de inserção, desta última são apresentados sete algoritmos.

Na Busca Tabu, são apresentadas sete estratégias, e a conclusão é que o desenvolvimento de heurísticas para o PRV tem sofrido bastante progresso desde o primeiro algoritmo proposto, sendo que a Busca Tabu mais eficiente proporciona resolver instâncias de tamanho médio de maneira ótima ou muito perto disso.

Em Desrochers *et al.* [8], é proposto um sistema que ajuda na classificação e na modelagem de um problema real, para uma grande variedade de classes do PRV, e na escolha de um algoritmo que pode ser aplicado ao modelo resultante do problema. A maioria dos modelos que foram considerados na literatura, segundo eles, podem ser classificados de acordo com o esquema proposto. Tal proposta surgiu pela motivação de o PRV possuir muitas variantes, e para cada uma, muitos algoritmos de resolução disponíveis, sendo as-

sim, é muito difícil selecionar um método que sirva muito bem para o problema específico. O conhecimento e a perícia que foram implantadas no sistema, foram aplicados em duas fases; a primeira para obter uma representação abstrata do problema em questão, e a segunda, para escolher dentre os vários algoritmos disponíveis, o mais apropriado para o modelo gerado. Tal esquema pode ser uma ferramenta útil em esclarecer as características do PRV prático, modelar problemas reais e trazer alguma uniformidade na literatura do assunto.

Dos problemas de PRVMD que serão apresentados resumidamente a seguir, dois deles ([9] e [10]) consideram a técnica de *backhauling*, que é uma maneira eficiente de reduzir custos de transporte através da economia de combustível. Nessa versão, cada cliente não só recebe mercadorias, como também pode despachar mercadorias através dos veículos para serem levadas ao depósito. Neste modelo, normalmente o veículo entrega todas as mercadorias e só depois cumpre uma rota de coleta de mercadorias que serão levadas ao depósito [10].

O algoritmo proposto por Renaud *et al.* [5] foi baseado na metaheurística busca tabu, inicialmente proposta por Glover [19]. Ele consiste em explorar o espaço de busca se movimentando de uma solução corrente s_c até o seu melhor vizinho s_v. Este movimento é feito mesmo que s_v tenha a avaliação pior do que s_c. Desse modo aumenta a probabilidade de sair de um ótimo local. Para evitar a formação de ciclos, soluções que foram recentemente visitadas são proibidas de serem visitadas novamente, ou declaradas *tabu* durante um determinado número de iterações. Estratégias de intensificação podem ser aplicadas para acentuar a busca em uma região promissora do espaço de solução. Em contrapartida, diversificação pode ser aplicada para possibilitar a busca em regiões menos exploradas. O algoritmo contém duas partes: construção da solução inicial e uma busca local.

O algoritmo proposto foi testado com 11 problemas clássicos da literatura propostos por Christofides e Eilon [20] e por Gillett e Johnson [21] e com mais 12 problemas de Chao *et al.* [22]. Os resultados obtidos mostraram que o algoritmo de Gillett e Johnson é claramente dominado pelo de Chao *et al.* e pelo proposto por [5]. Entre os algoritmos citados, o algoritmo de [5] mostrou superioridade em quase todos os casos.

Uma das primeiras heurísticas para resolver o PRVMD foi proposta por Tillman [6]

e usa o critério de economia de Clarke e Wright [23]. Primeiramente, o algoritmo aloca cada cliente ao depósito localizado mais próximo a este, e depois, constrói rotas para cada depósito. Essas rotas são gradualmente combinadas usando um critério de economia que leva em conta a presença de depósitos diferentes, resultando assim em rotas maiores. Essa heurística é apenas construtiva e não possui uma fase posterior de refinamento.

A heurística de construção proposta por Wren and Holliday [7] propõe um método de varredura que pode ser aplicado para um PRV ou um PRVMD. A heurística, primeiramente, aloca de forma temporária cada cidade ao depósito que se localiza mais próximo desta e, usando um eixo de referência, computa o seu ângulo polar em relação ao seu depósito. Depois todas as cidades são ordenadas de forma ascendente de acordo com seu ângulo polar. Elas são então assinaladas de forma iterativa a qualquer outra rota, nova ou existente, que tiver a menor distância adicional. Realocações a depósitos diferentes são então feitas.

Outra forma de calcular o ângulo polar proposta pelos autores leva em conta a configuração dos pontos que rodeiam cada depósito. Essa nova ordenação é usada para gerar quatro soluções iniciais diferentes. A melhor delas é escolhida como entrada para uma fase de aperfeiçoamento. Esta fase usa sete procedimentos repetidamente até não haver mais melhora.

A diferença entre o problema de Salhi e Nagy [9] para o de Min *et al.* [10], que serão expostos a seguir, é que apesar de ambos serem com *backhauling*, no de Min *et al.* [10] um veículo só coleta mercadorias depois de já haver feito a entrega de todos os clientes, ou seja, depois de estar vazio.

O objetvo de Salhi e Nagy [9] foi desenvolver uma heurística de inserção para o PRV com *backhauling* adaptada para o caso de múltiplos depósitos. Ele apresenta algumas heurísticas de inserção que extendem a proposta de Casco *et al.* [24] para inserções por agrupamento (*cluster*). A heurística base é a heurística proposta por Casco *et al.* [24] com um aprimoramento da fórmula de avaliação de custo. O problema é solucionado, primeiramente, para um PRV, considerando apenas os clientes que recebem mercadoria. Depois, os clientes que enviam mercadorias são inseridos um a um, usando o cálculo de menor custo, até que não haja mais nenhum que não tenha sido inserido.

As propostas foram testadas com casos da literatura e casos gerados pelos próprios autores, que variam de 50 à 249 clientes e de 2 à 5 depósitos. No total foram 25 casos de teste. Os resultados obtidos foram bons tanto para o caso simples do PRV quanto para o caso de múltiplos depósitos, com um tempo de execução relativamente baixo.

Já em Min *et al.* [10] foi desenvolvido um modelo matemático que objetiva responder basicamente três questões: 1) quais pontos de coleta (clientes que enviam mercadorias) devem ser incluídos em uma dada rota, 2) quais caminhões devem ser alocados para quais clientes de coleta e de entrega de mercadorias, e 3) em que sequência os clientes que enviam e recebem mercadorias devem ser atendidos.

O procedimento proposto foi dividido em três fases: 1) alocação dos clientes e vendedores para grupos com capacidade limitada, 2) associação de clientes e vendedores a depósitos e rotas, e 3) configuração individual de rota, considerando a capacidade tanto dos veículos quanto dos depósitos. Estes são executados em sequência com as saídas de um procedimento servindo de entrada para o seguinte.

Os problemas usados para os testes foram gerados em cima de problemas reais com pequenas alterações para manter o sigilo das informações. Os resultados obtidos demonstraram que a metodologia proposta pode resolver de maneira prática problemas de roteamento de múltiplos depósitos e múltiplos veículos que incorporem capacidades de veículos e de centros de distribuição assim como os que incluem *backhauling*.

Capítulo 3

A metaheurística GRASP

Metaheurística são heurísticas genéricas para a solução aproximada principalmente de problemas de otimização combinatória de elevada complexidade computacional. Entre elas, podemos destacar: busca tabu, GRASP, algoritmos genéticos, VNS, colônias de formigas e outras mais.

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) proposta por Feo e Resende [25] é um processo iterativo do tipo *multi-start* para obter soluções aproximadas, eventualmente ótimas, para problemas de otimização combinatória. Suas iterações consistem, basicamente, de duas fases: uma de construção e outra de busca local. A primeira fase tem como objetivo gerar uma solução viável para o problema proposto. Esta solução é construída iterativamente elemento a elemento e posteriormente, na fase de busca local, é aprimorada com buscas nas suas vizinhanças até que um ótimo local seja encontrado. Ao final da execução, temos como resultado a melhor solução encontrada ao longo de todas as iterações do GRASP.

Um pseudo-código genérico do GRASP é ilustrado na Figura 3.1 para um problema de minimização. Na linha 1 do pseudo-código a melhor solução e a sua função objetivo são inicializadas. Na linha 2 são lidos os dados de entrada do problema. As iterações GRASP são realizadas das linhas 3 à 9 e terminadas após realizadas *MaxIter* iterações. Na linha 4 é executada a fase de construção, enquanto na linha 5 é a fase de busca local. Na linha 6 é feita a verificação se a solução obtida pela iteração é melhor do que a melhor solução encontrada até o momento; em caso positivo, na linha 7 é feita a atualização da melhor solução e da sua função de avaliação.

A cada iteração da fase de construção, o conjunto de elementos candidatos é formado

```
procedure GRASP;
1.   f* ← ∞ ; x* ← Ø ;
2.   Ler os dados do problema;
3.   Para k = 1,...,MaxIter Faça
4.        Construir uma solução randomizada x (fase de construção);
5.        Encontrar y aplicando busca local a x (fase de busca local);
6.        Se f(y) < f* Então
7.             x* ← y ; f* ← f(y);
8.        Fim_Se
9.   Fim_Para
10.  retornar x*
Fim GRASP
```

Figura 3.1: Pseudo-Código da versão básica da metaheurística GRASP.

por todos os elementos que ainda não foram incorporados à solução parcial em construção e que não inviabilizam a solução caso venham a ser incorporados. A escolha do próximo elemento a ser incorporado é determinada pela avaliação de todos os elementos candidatos de acordo com uma função gulosa. Esta função avalia os benefícios ganhos com a inserção deste elemento na solução em construção (custo incremental). A avaliação dos elementos, segundo esta função, leva à criação de uma lista restrita de candidatos (LRC) formada por um subconjunto dos melhores candidatos, isto é, aqueles cuja incorporação à solução parcial corrente resulta nos menores custos incrementais (aspecto guloso do algoritmo). O elemento a ser incorporado à solução parcial é selecionado aleatoriamente dentre aqueles da LRC (aspecto probabilístico do algoritmo). Uma vez que o elemento selecionado foi incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados (aspecto adaptativo do algoritmo) [26].

Segundo [27], existem duas estratégias básicas usadas para construir a LRC: pelo número de elementos (percentual do tamanho da lista de candidatos) ou pela qualidade dos elementos.

Na primeira estratégia, que é utilizada nesta dissertação, os elementos candidatos são ordenados de forma decrescente de benefício segundo a função de avaliação f e os p primeiros elementos são incluídos na LRC. O valor de p é definido na equação 3.1.

$$p = 1 + \alpha(a - 1) \tag{3.1}$$

Onde a é número total de candidatos e α é um parâmetro que tem valores definidos no intervalo $[0, 1]$). Note que, se $\alpha = 0$ um algoritmo totalmente guloso é executado, já que só é possível à escolha de um único elemento a ser inserido na solução. Por outro lado, se $\alpha = 1$, a lista conterá todos os possíveis candidatos, o que resultará em um algoritmo aleatório.

Na segunda estratégia, considere um problema de minimização e e_{min} o candidato de menor incremento para a solução parcial de acordo com a função f e e_{max}, o de maior incremento dentre todos os elementos candidatos. Os elementos candidatos de LRC serão formados por aqueles cujo valor retornado pela função f estiver no intervalo $[e_{min}, (1 - \alpha)(e_{max} - e_{min}) + e_{min}]$.

Como já citado, α é um valor definido no intervalo $[0, 1]$. Para $\alpha = 1$, estará sendo executado um algoritmo absolutamente guloso, já que somente os elementos de menor incremento e_{min} podem ser inseridos na solução e se $\alpha = 0$, a lista conterá todos os possíveis candidatos, o que resultará em um algoritmo aleatório.

A Figura 3.2 ilustra o pseudo-código da fase de construção do GRASP. A solução que será construída é inicializada na linha 1 do pseudo-código. A repetição que vai da linha 2 à 7 é feita até que a solução completa seja construída. Na linha 3 os custos incrementais dos elementos candidatos são avaliados e na 4 é formada a LRC. Um elemento da LRC é selecionado aleatoriamente na linha 5, observe que esta escolha aleatória permite que soluções distintas sejam geradas a cada execução deste algoritmo de construção, e na linha 6 este elemento é adicionado à solução.

```
procedure Construtivo;
1.   x ← ∅ ;
2.   Enquanto x não for uma solução completa Faça
3.        Avaliar os custos dos elementos candidatos;
4.        Construir a lista restrita de candidatos LRC;
5.        Selecionar aleatoriamente um elemento s dentre os da LRC;
6.        x ← x ∪ {s};
7.   Fim_Enquanto;
8.   retorna x;
Fim Construtivo;
```

Figura 3.2: Pseudo-Código da fase de construção.

No caso do GRASP, como no caso de muitos métodos construtivos, a solução gerada pela fase de construção do GRASP não representa necessariamente um ótimo local, no que diz respeito a definições de vizinhança simples. Então, é sempre benéfico aplicar uma busca local na tentativa de melhorar a solução inicial. A busca local trabalha de um modo iterativo, substituindo sucessivamente a solução corrente por uma solução melhor pertencente à vizinhança da solução corrente. A busca local termina quando nenhuma solução melhor for encontrada na vizinhança.

Uma estrutura de vizinhança N para um problema P relaciona a solução x do problema a um subconjunto de soluções $N(x)$. Uma solução x é considerada *ótimo local* segundo $N(x)$ se não existir melhor solução que x em $N(x)$. Dada uma estrutura de vizinhança N, um pseudo-código de um algoritmo de busca local geral é apresentado na Figura 3.3. Dada uma solução inicial x, que foi gerada pela fase de construção do algoritmo GRASP, uma repetição é feita da linha 1 à linha 4 até que nenhuma solução melhor que a solução corrente (x) seja encontrada na vizinhança N. Na linha 2 tenta-se obter uma solução $x^{'}$ da vizinhança N melhor que a solução corrente x. Na linha 3 a solução corrente passa a ser a melhor solução. Na linha 5 a melhor solução da vizinhança N em relação à solução x inicial é retornada.

A chave do sucesso do algoritmo de busca local inclui a escolha de uma estrutura de vizinhança adequada, com técnicas eficientes e uma boa solução de início.

```
procedure Busca_Local;
1.  Enquanto x não for uma solução localmente ótima Faça
2.        Obter x' ∈ N(x) tal que f(x') < f(x);
3.        x ← x';
4.  Fim_Enquanto;
5.  retorna x;
Fim Busca_Local;
```

Figura 3.3: Pseudo-Código da fase de busca local.

Existem duas maneiras de implementar a busca local: a melhor alteração (*best improving*) e a primeira melhora (*first improving*). No primeiro caso, toda a vizinhança da solução é percorrida e somente a melhor alteração é feita. Já no segundo, a primeira alteração que proporcionar melhora é executada, atualizando-se a solução semente e a partir desta, uma nova vizinhança é formada.

Vários conceitos e módulos de aperfeiçoamento têm sido propostos para melhorar as soluções da metaheurística GRASP. Um sumário sobre as principais contribuições, tais como: reconexão por caminhos (*Path Relinking)*[28], GRASP Reativo [29], Memória a Longo Prazo [30] e Filtro [31] é feito a seguir.

Reconexão Por Caminhos

O procedimento de reconexão por caminhos (RC) foi proposto originalmente para os métodos de Busca Tabu (*Tabu Search*) e *Scatter Search* [32] como uma estratégia de intensificação, explorando trajetórias que conectavam soluções de boa qualidade (soluções elite) obtidas pela heurística. Neste contexto, na busca por melhores soluções são gerados e explorados caminhos no espaço de soluções partindo de uma ou mais soluções elite e levando a outras soluções de boa qualidade. Isto é alcançado selecionando-se movimentos que introduzam atributos das *soluções alvo* na *solução base*. Esta técnica pode ser vista como uma estratégia que busca incorporar atributos das soluções de boa qualidade, favorecendo a seleção de movimentos que os contenham [26].

Este procedimento consiste em analisar todas as soluções intermediárias entre duas soluções de boa qualidade, procurando, com isso, encontrar uma terceira que seja melhor que as soluções extremos. Tendo sido armazenado, no decorrer da execução da heurística considerada, o conjunto elite, composto pelas p melhores soluções tais que estas sejam distintas entre si, as soluções extremos são a *solução base* (sb) e a *solução alvo* ((sa)).

O algoritmo RC computa a diferença simétrica entre sb e sa (*dif(sa,sb)*), resultando no conjunto de movimentos que devem ser aplicados a sb para alcançar a sa. A partir da sb, o melhor movimento ainda não executado de *dif(sa,sb)* é aplicado à sb, até que sa seja atingida. A melhor solução encontrada ao longo desta trajetória é considerada como candidata à inserção no conjunto elite e a melhor solução globalmente já encontrada é atualizada quando for o caso [26].

A RC pode analisar as soluções intermediárias entre a *solução base* e uma *solução alvo* em um ou dois sentidos, ou seja, fazendo com que a sb sofra as alterações necessárias para alcançar a sa ou vice e versa [28]. Uma ilustração do módulo de RC é apresentada na figura 3.4. Por esta figura percebe-se que a busca é mais intensa próximo a sb e menos intensa quando se aproxima de sa.

Desta forma costuma-se escolher como sb a melhor das duas soluções extremas consideradas, caso somente um sentido (sb) -> (sa) seja implementado.

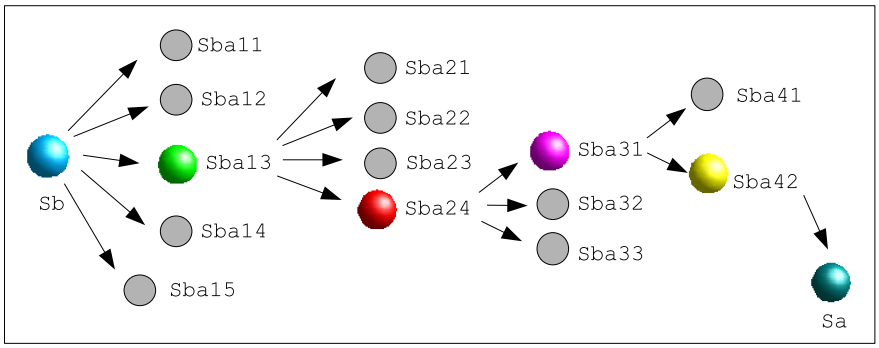

Figura 3.4: Uma ilustração dos passos de um módulo de RC de uma solução sb para uma solução sa.

No exemplo da figura 3.4 temos, ao dar o primeiro passo para percorrer o caminho da solução S_b para a solução S_a, cinco opções possíveis: S_{ba11}, S_{ba12}, S_{ba13}, S_{ba14} e S_{ba15}. Sendo S_{ba13} a que gera a melhor solução dentre as opções do primeiro passo, tomamos esta como intermediária e a partir desta passamos para o segundo passo vendo todas as possibilidades, tomando a melhor opção como intermediária e seguindo daí em diante até chegar na solução S_a.

GRASP Reativo

O valor do parâmetro α, que controla o nível de aleatorização da fase de construção do GRASP, é importantíssimo para um bom desempenho da heurística. Não existe um melhor valor para α que se adeque para todos os casos e problemas, normalmente este depende de características particulares do problema, características e tamanhos das instâncias entre outros fatores.

Um estratégia possível para incorporar mecanismos de aprendizado na fase de construção é o procedimento GRASP reativo proposto por Prais e Ribeiro [29]. Neste caso, o valor do parâmetro α que define a LRC não é fixo. Em vez disto, esse valor é selecionado a cada iteração dentre os elementos de uma distribuição discreta de valores possíveis. O melhor valor para α é determinado dinamicamente, fazendo com que cada α tenha uma probabilidade de ser escolhido de acordo com a qualidade das soluções geradas por este na fase de construção, de maneira adaptativa.

Neste método existe um conjuntos de n valores pré-definidos para α. Quando o construtivo é executado, um dos valores de α é sorteado de maneira aleatória e usado para construir a solução base. A princípio, cada valor possui a mesma probabilidade $\frac{1}{n}$ de ser sorteado, e então vai sendo feita uma atualização periódica dessas probabilidades de acordo com a qualidade das soluções que foram geradas usando cada um dos valores pré-definidos de α [29, 33].

Memória a Longo Prazo

O GRASP em sua proposta original [25] é composto de iterações independentes, ou seja, não possui nenhum método de armazenamento de características de soluções geradas em iterações anteriores que possam melhor guiar o processo de geração de soluções futuras, para que estas sejam mais promissoras. Por isso, Fleurent e Glover [30] propuseram um esquema para tratar esta questão em heurísticas do tipo *multi-start*, tornando-as adaptativas.

A proposta foi manter um conjunto elite de soluções para ser usado na fase de construção. Para tornar-se uma solução elite, uma solução corrente deve ser melhor do que a pior solução do conjunto, e também suficientemente diferente das demais soluções elite.

A partir daí, com o conjunto elite formado, as soluções das iterações seguintes podem ser formadas tendo como base informações relevantes extraídas das soluções de boa qualidade existentes no conjunto elite. Este procedimento pode ser usado durante algumas iterações seguintes do GRASP. Em seguida, retorna-se ao método original de construção do GRASP. Estas duas formas de construir soluções podem ser usadas alternadamente nas iterações restantes do GRASP cujo balanceamento é definido *a priori*.

GRASP com Filtro

Os métodos de construção em otimização combinatória, normalmente, têm a função de gerar uma solução viável para o problema. Métodos que utilizam construção e busca local se caracterizam por construir e posteriormente tentar refinar a solução inicial.

O que a literatura mostra é que, normalmente, os métodos de refinamento (busca local) dominam os tempos computacionais exigidos, ou seja, estes utilizam a maior parte dos tempos totais do método.

Uma forma de tentar reduzir os tempos de uma busca local é oferecer a estes procedimentos soluções iniciais de melhor qualidade. Esta sugestão é bem justificada no caso da heurística GRASP, onde a cada iteração é gerada uma solução inicial e posteriormente essa é refinada por uma busca local.

A proposta de usar um filtro na etapa de construção do GRASP, simplesmente, se reduz a cada iteração GRASP gerar p soluções iniciais e selecionar somente a melhor delas para efetuar a busca local.

Contudo, este procedimento tende a aumentar o risco de efetuar busca local em soluções analisadas em iterações passadas.

Portanto é conveniente adotar algum mecanismo de memória para tentar evitar esta situação.

No caso do GRASP esta proposta é válida, pois os tempos exigidos na fase de construção são muito menores do que os tempos exigidos pela fase de busca local [31].

Capítulo 4

Propostas

Este trabalho apresenta uma formulação matemática do PRSM descrevendo-o como um Problema de Programação Linear Inteira (PPLI), que utiliza variáveis de fluxo para eliminar a formação de soluções desconexas da origem. Esta formulação foi proposta para obter soluções exatas para instâncias de pequeno porte, de modo a possibilitar uma comparação das soluções aproximadas obtidas pelo GRASP com as soluções exatas, ao menos neste tipo de instâncias.

Este trabalho apresenta também diferentes versões da metaheurística GRASP aplicadas à solução do problema de gerar percursos otimizados para um problema real de Roteamento de Sondas de Manutenção (PRSM), cuja meta é alocar as sondas a um conjunto de poços petrolíferos terrestres da região nordeste do Brasil. O objetivo principal é o de analisar a importância dos algoritmos de construção e busca local, incluindo a reconexão por caminhos, filtro e versões adaptativas no desempenho da metaheurística GRASP.

As propostas iniciais consistem em dois algoritmos construtivos e três algoritmos de busca local. Cada combinação: *construtivo + busca local* gerou uma versão GRASP, totalizando, em uma primeira etapa, seis versões que são definidas como GRASP puro. Numa segunda etapa foi proposto um módulo de reconexão por caminhos (*path relinking*) incorporado a algumas versões de GRASP puro. Em uma terceira etapa propusemos outras estratégias de GRASP adaptativo. Posteriormente, em uma quarta etapa, com as melhores versões obtidas considerando as três etapas anteriores, introduzimos um módulo de filtro em algumas delas.

A implementação neste trabalho se deu na linguagem de programação C, usando o

compilador *GNY gcc* em sua versão 3.2 em ambiente operacional Linux.

4.1 Formulação Matemática Proposta

O PRSM pertence à classe dos modelos de Problema de Roteamento de uma frota de veículos com múltiplos depósitos (PRVMD), problema este classificado como NP-difícil [34]. O PRSM, por sua vez, também é classificado como NP-difícil [4], limitando com isso o uso de métodos exatos para a solução do problema de maneira viável. Entretanto, mesmo com estas limitações, é reconhecida a importância de desenvolver uma formulação matemática destes problemas, tanto para avaliar os resultados de métodos aproximados em instâncias pequenas, como para se obter limites inferiores de qualidade para o valor ótimo (valor ou função objetivo associado à solução ótima). Desta forma, propomos nesta seção uma formulação matemática do PRSM, modelando-o como um problema de programação inteira com restrições lineares e não lineares. Portanto, considere a seguinte notação:

Admitamos que n poços são indexados por $N = \{1, 2, ..., n\}$, que o conjunto de poços é definido como $P = \{p_1, p_2, ..., p_n\}$, que k sondas são indexadas por $K = \{1, 2, ..., k\}$ e que t_{ij} é o tempo gasto para percorrer a distância entre os poços p_i e p_j, com $i \neq j$ e p_i, $p_j \in P$. Cada poço p_i possui uma vazão $v(i)$ por unidade de tempo e um tempo de serviço $ts(i)$ associado. No início da rota, cada sonda s_k está localizada em algum poço atendido no período anterior e não previsto para este período de atendimento, sendo este um dado de entrada. Cada poço deve ser visitado uma única vez e por uma única sonda e nenhum poço pode ficar sem atendimento. Assim, temos as seguintes constantes e variáveis empregadas neste modelo:

Constantes

- $N = \{1, 2, ..., n\}$: Conjunto de poços que estão a espera de algum serviço no atual período de planejamento.
- $K = \{1, 2, ..., k\}$: Conjunto de sondas que estão disponíveis para atender os poços que requisitaram serviço no atual período de planejamento.
- $S = \{O_1, O_2, ..., O_k\}$: Conjunto das origens das k sondas disponíveis no atual período

de planejamento, onde O_k é um inteiro que representa a origem da sonda k.

- $\bar{N} = N \cup S$: União do conjunto das origens das sondas com o conjunto dos poços que estão à espera de serviço.
- t_{ij}: Tempo estimado para transportar uma sonda do poço p_i ao poço p_j.
- ts_i: Tempo estimado para realizar o serviço requisitado pelo poço p_i, contando a partir da chegada da sonda em p_i.
- v_i: Vazão do poço p_i por unidade de tempo.
- HOR: Período em unidades de tempo associado a um horizonte de planejamento.
- $N_i \subseteq N$: Subconjunto de poços de N que podem ser atendidos pela sonda s_i, onde $i \in K$.
- $PRAZO$: Tempo limite máximo para realizar o serviço de um poço (contabilizado a partir do seu pedido).
- $ATRASO(p_i)$: Tempo em que o poço p_i ficou sem atendimento no período anterior (caso onde o poço p_i não pode ser atendido no período anterior).
- $FOLGA(p_i) = PRAZO - ATRASO(p_i)$: Tempo restante para realizar o serviço do poço p_i.

Variáveis

- $y_i \geq 0$ e inteiro: Tempo em que o poço p_i ficou parado no atual período de planejamento, onde $y_i = 0, \forall i \in S$

$$x_{ij}^k : \begin{cases} 1 & \text{, se a sonda } k \text{ atende o poço } p_j \text{ logo após o } p_i. \\ 0 & \text{, caso contrário.} \end{cases}$$

$x^k{}_{ij} \in \{0, 1\}, \forall\ i \in \bar{N}, \forall\ j \in N\ e\ \forall\ k \in K$

Minimizar:

$$\sum_{i \in N} v_i y_i \tag{4.1}$$

Sujeito à:

$$y_j \leq FOLGA(p_j), \forall\ j \in N \tag{4.2}$$

$$\sum_{k \in K} \sum_{i \in \bar{N}} x_{ij}^k\ =\ 1, \forall\ j \in N, \text{ onde } i \neq j \tag{4.3}$$

$$\sum_{j \in N} \sum_{k \in K} x_{ij}^k \leq 1, \forall\ i \in \bar{N}, \text{ onde } i \neq j \tag{4.4}$$

$$\sum_{i \in \bar{N}} x_{ij}^k\ -\ \sum_{i \in N} x_{ji}^k \geq 0, \forall\ k \in K, \forall\ j \in N, \text{ onde } i \neq j \tag{4.5}$$

$$\sum_{j \in N} x_{ij}^k\ =\ 1, \text{ onde } i = O_k, \forall\ k \in K \tag{4.6}$$

$$\sum_{i \in S} \sum_{j \in N} x_{ij}^k\ =\ 1, \forall\ k \in K \tag{4.7}$$

$$y_j \geq y_i\ x_{ij}^k + t_{ij}\ x_{ij}^k + ts_j\ x_{ij}^k, \forall\ i \in \bar{N}, \forall\ j \in N(i \neq j), \forall\ k \in K \tag{4.8}$$

$$\sum_{i \in \bar{N}} \sum_{j \in N} (t_{ij}\ x_{ij}^k\ +\ ts_j\ x_{ij}^k) \leq HOR, \forall\ k \in K, i \neq j \tag{4.9}$$

$$y_j \geq 0 \text{ é inteiro, } j \in N \tag{4.10}$$

$$x_{ij}^k \in \{0, 1\}, \forall\ i \in \bar{N}, \forall\ j \in N(i \neq j)\ e\ \forall\ k \in K \tag{4.11}$$

Na formulação descrita por (4.1) a (4.11), a função objetivo 4.1 requer a melhor alocação dos poços ao conjunto das sondas de modo que a quantidade total de óleo que deixou de ser coletado seja minimizado. As restrições 4.2 exigem que cada poço com pedido de manutenção seja atendido dentro dos prazos estipulados, ou seja, em um tempo menor ou igual ao tempo máximo previamente definido.

As igualdades 4.3 obrigam que cada poço $j \in N$ seja visitado apenas uma única vez e por uma única sonda dentro do atual horizonte de planejamento. As restrições 4.4 impõem que, para cada poço, exista no máximo uma única sonda partindo do mesmo rumo a outro poço. O motivo de não usar uma igualdade é que, se o poço for o último a ser visitado por uma sonda, esta permanece neste local até o final do atual período. As restrições 4.5 exigem que quando uma sonda chegar em um poço p_j, somente esta sonda poderá partir deste mesmo poço. As restrições 4.6 obrigam que cada sonda saia exatamente uma única

vez de sua origem na sua lista de atendimento, e 4.7 que cada sonda s_k saia exatamente uma única vez da sua origem. Assim, a combinação das restrições 4.6 e 4.7 fixa a origem da lista de atendimento de cada sonda. As desigualdades 4.8 indicam que o tempo que um poço p_j fica parado é no mínimo igual à soma: tempo em que o poço p_i antecessor imediato de p_j na rota ficou parado (y_i) + tempo para transportar a sonda do poço p_i ao poço p_j + tempo estimado para o conserto do poço p_j. As restrições 4.9 exigem que a quantidade de tempo gasta por cada sonda s_k onde $k \in K$ no período seja menor ou igual ao horizonte de planejamento. Finalmente 4.10 e 4.11 são as variáveis do problema.

A equação 4.8 leva a não-linearidade da função objetivo dada pela equação 4.1. Uma forma de linearizar estas restrições é proposta a seguir.

4.1.1 Proposta de Linearização da Formulação Matemática

As restrições 4.8 podem ser substituídas por restrições equivalentes lineares, descritas a seguir:

$(y_i + t_{ij}\ x_{ij}^k\ +\ ts_j\ x_{ij}^k) - y_j \leq (1 - x_{ij}^k)M$, onde $M \gg 0$, $\forall\ i \in \bar{N}, \forall\ j \in N (i \neq j)$ e $\forall\ k \in K$ (4.19)

As restrições 4.19 devem ser analisadas em duas situações:

1) Quando $x_{ij}^k = 0$, neste caso 4.19 se reduz a: $(y_i \leq y_j + M)$, onde $M \gg 0$

2) Quando $x_{ij}^k = 1$, neste caso 4.19 será da forma:

$(y_i + t_{ij} + ts_j) - y_j \leq 0$ (que é a condição a ser satisfeita quando uma mesma sonda s_k atende os poços p_i e p_j consecutivamente.

Assim, reescrevemos a seguir a formulação matemática proposta, mas agora de forma linearizada:

Minimizar:

$$\sum_{i \in N} v_i y_i \tag{4.12}$$

Sujeito à:

$$y_i \leq FOLGA(p_j), \forall\ j \in N \tag{4.13}$$

$$\sum_{k \in K} \sum_{i \in \bar{N}} x_{ij}^k\ =\ 1, \forall\ j \in N, \text{ onde } i \neq j \tag{4.14}$$

$$\sum_{j \in N} \sum_{k \in K} x_{ij}^k \leq 1, \forall\ i \in \bar{N}, \text{ onde } i \neq j \tag{4.15}$$

$$\sum_{i \in \bar{N}} x_{ij}^k\ -\ \sum_{i \in N} x_{ji}^k \geq 0, \forall\ k \in K, \forall\ j \in N, \text{ onde } i \neq j \tag{4.16}$$

$$\sum_{j \in N} x_{ij}^k\ =\ 1, \text{ onde } i = O_k, \forall\ k \in K \tag{4.17}$$

$$\sum_{i \in S} \sum_{j \in N} x_{ij}^k\ =\ 1, \forall\ k \in K \tag{4.18}$$

$$(y_i\ +\ t_{ij}\ x_{ij}^k\ +\ ts_j\ x_{ij}^k)\ -\ y_j \leq (1\ -\ x_{ij}^k)M, \quad \forall\ i \in \bar{N}, \forall\ j \in N (i \neq j)\ e\ \forall\ k \in K \tag{4.19}$$

$$\sum_{i \in \bar{N}} \sum_{j \in N} (t_{ij}\ x_{ij}^k\ +\ ts_j\ x_{ij}^k) \leq HOR, \forall\ k \in K, i \neq j \tag{4.20}$$

$$y_j \geq 0 \text{ e inteiro, } j \in N \tag{4.21}$$

$$x_{ij}^k \in \{0, 1\}, \forall\ iin\bar{N}, \forall\ j \in N (i \neq j)\ e\ \forall\ k \in K \tag{4.22}$$

4.2 Construtivo C1

Esta heurística construtiva é baseada no conceito de inserção do vizinho mais próximo. Para isso, inicialmente é definida uma fórmula para determinar o nível de prioridade de cada poço a ser usada no processo de seleção do próximo poço a ser inserido numa rota parcial.

Considere R_k uma rota em construção, sempre iniciando num poço origem (poço onde a sonda associada s_k está localizada no início do planejamento) e seja p_i, o poço inserido em R_k mais recentemente. Então para todo poço p_j ainda não alocado, o nível de prioridade de p_j para a rota da sonda s_k é dado por $pri(p_j, s_k) = [v_j]/[t_{ij} + ts_j]$; onde v_j é a vazão por unidade de tempo do poço p_j, t_{ij} é o tempo para ir do poço p_i até o poço

p_j, onde p_i é o último inserido na rota de s_k, e ts_j é o tempo estimado para a realização do serviço requisitado pelo poço p_j a partir da chegada da sonda.

É apresentado na figura 4.1 um pseudo-código do procedimento C1. Definido o cálculo de prioridade, é executado um *loop* da linha 1 à linha 9 enquanto existir algum poço sem previsão de atendimento. Na linha 2 é selecionada a sonda s_k que irá receber o poço sorteado. Esse algoritmo possui uma lista de sondas que é percorrida seqüencialmente de forma circular e a sonda selecionada é a próxima sonda da lista. Na linha 3, é montada a Lista Restrita de Candidatos (LRC) da próxima inserção de poços na sonda s_k selecionada, esta lista é criada a partir de uma lista formada por todos os poços ainda não alocados, ordenados de maneira decrescente de prioridade com base na localização do último poço alocado para a sonda s_k. A cardinalidade da LRC é dada pelo parâmetro α descrito no capítulo 3. Na linha 4 um poço p_i pertencente à LRC é selecionado aleatoriamente e adicionado na seqüência de atendimento da sonda s_k na linha 5. Nas linhas 6 e 7, respectivamente, a vazão perdida e o tempo gasto pela sonda s_k são atualizados. Na linha 8, a lista de poços ainda não alocados é atualizada (retirando o poço p_i), e a seguir, partimos para selecionar o próximo poço da próxima sonda.

Esta escolha aleatória possibilita que diferentes soluções sejam geradas por este procedimento.

Observe que esta forma de construção se adapta às características da etapa construtiva do GRASP, ou seja, um construtivo: iterativo (os poços são inseridos um a um), adaptativo pois no processo de inserção de um novo poço, a composição da LRC é dependente das escolhas anteriores, aleatória pois nem sempre o melhor candidato de uma LRC é escolhida, e gulosa, pois se tomarmos LRC composto de um único elemento, o melhor candidato será sempre selecionado. O procedimento de alocar um poço para cada sonda de forma alternada é justificado para se obter um balanceamento de carga entre as sondas e, com isso, tentar otimizar o objetivo de minimizar a quantidade de óleo que deixou de ser coletado no período considerado.

```
procedure Construtivo_1
1.  Enquanto existir poço sem atendimento faça
2.          k  = próxima sonda;
3.          LRC = constroi_LRC(sonda[k], alfa);
4.          poço_escolhido = sorteio_poço(LRC);
5.          ListaAtendimento(sonda[k]) = poço_escolhido;
6.          AtualizaVazaoPerdida(sonda[k]);
7.          AtualizaTempoGasto(sonda[k]);
8.          AtualizaListaPoços(poço_escolhido);
9 .  Fim Enquanto;
10.  retorna solução;
Fim Construtivo_1
```

Figura 4.1: Pseudo-código do Construtivo 1

4.3 Construtivo C2

Neste construtivo, descrito na figura 4.2, colocamos como prioridade apenas o item vazão de cada poço. Para isso, é gerada uma única lista de poços com todos os poços que ainda não foram alocados para nenhuma das sondas (lista de poços), ordenada de forma decrescente de acordo com a vazão desses poços, como podemos ver na linha 1. Então é feito um *loop* até que todos os poços que estão na fila a espera de atendimento sejam alocados, demonstrado da linha 2 à linha 10. Assim, a cada etapa de inserção de um novo poço, uma Lista Restrita de Candidato (LRC) é formada com os w primeiros poços da Lista de Poços. O tamanho de LRC é calculado com base no dado de entrada α como no Construtivo 1, observado na linha 3. A seguir é sorteado um dos poços de LRC (linha 4), mas ao contrário do Construtivo 1, esse poço é alocado para a sonda que primeiro puder atendê-lo (linha 5), pois assim, estaremos minimizando a perda de vazão desse poço que é um dos w poços remanescentes de maior vazão.

A sonda para a qual o poço será alocado é selecionada da seguinte maneira. Para cada sonda é somado o tempo gasto por esta desde o início do período de planejamento para atender todos os poços já alocados a esta sonda e que estão em sua lista de atendimento (tempo de deslocamento da sonda mais o tempo gasto para a realização dos serviços previstos pelos poços da lista) mais a distância que a sonda se encontra do poço sorteado. A sonda que fornecer o menor tempo acumulado é a sonda escolhida para atender o poço sorteado.

Posteriormente, a vazão perdida e o tempo gasto da sonda escolhida são atualizados (linhas 7 e 8 respectivamente) e então a lista de poços é atualizada, retirando o poço que foi selecionado (linha 9).

```
procedure Construtivo_2
1.  Lista_Poços = Poços_Ordenado_Vazão_Decrescente;
2.  Enquanto existir poço sem atendimento faça
3.          LRC = constroi_LRC(alfa);
4.          poço_escolhido = sorteio_poço(LRC);
5.          sonda_escolhida = Sonda_Que_Primeiro_Atende(poço_escolhido);
6.          ListaAtendimento(sonda_escolhida) = poço_escolhido;
7.          AtualizaVazaoPerdida(sonda_escolhida);
8.          AtualizaTempoGasto(sonda_escolhida);
9.          AtualizaListaPoços(poço_escolhido);
10. Fim Enquanto;
11. retorna solução;
Fim Construtivo_2
```

Figura 4.2: Pseudo-código do Construtivo 2

4.4 Busca Local BL1

Essa primeira busca local proposta, descrita no pseudo-código da figura 4.3, trabalha com duas estruturas de vizinhança que são executadas alternadamente em *loop* (representado da linha 2 à linha 10 do pseudo-código), até que não haja melhora em nenhuma delas.

Antes de começar a busca, como visto na linha 1 do algoritmo, uma estrutura com os *r-vizinhos* mais próximos de cada poço é montada e passada para os procedimentos de vizinhança, onde para cada poço p_i, existe nesta estrutura os r vizinhos mais próximos deste poço (onde r é um parâmetro de entrada).

Na linha 3, a variável *melhorou* é inicializada com o valor *zero*, pois ela comanda o *loop* externo (linhas 2 à 10), indicando quando as vizinhanças não produzem mais melhora. Nesse caso a variável *melhorou* não muda seu valor, o que indicará o fim do *loop*.

Da linha 4 à linha 7, um outro *loop* é executado. Este loop executa a *vizinhança 1* (*V1* representada na linha 5) para cada sonda, retornando a sonda corrente atualizada, quando for o caso, e o indicativo se houve melhora em *V1*.

Nas linhas 5 e 8, a variável *var_melhorou*, que é passada como parâmetro para as

vizinhanças, vai retornar 1 caso o procedimento retorne uma solução melhor, e 0 caso retorne a mesma solução (caso em que não houve melhora). Assim, no final da execução das duas vizinhanças, a variável *melhorou*, será *maior que zero* caso alguma das duas vizinhanças proporcione melhora, e *zero* caso contrário. Esta variável é atualizada nas linhas 6 e 9 de acordo com a melhora ou não nas vizinhanças.

```
 procedure BL_1
1.   Vizinhos_Mais_Proximos (r);
2.   Faça
3.          melhorou = 0;
4.          Para i variando de 1 até num_sondas faça
5.                 solução_base = Vizinhança1( i, solução_base, var_melhorou );
6.                 melhorou = melhorou + var_melhorou;
7.          Fim Para;
8.          solução_base = Vizinhança2( solução_base, var_melhorou );
9.          melhorou = melhorou + var_melhorou;
10.  Enquanto melhorou > 0;
11.  retorna solução;
 Fim BL_1
```

Figura 4.3: Pseudo-código da Busca Local 1

Na primeira vizinhança (*V1*), a partir de uma solução inicial, um poço é permutado sequencialmente com todos os poços que estão dentre seus *r-vizinhos* mais próximos e que se encontram na mesma sonda que ele, mantendo-se, ao final de todas as trocas, aquela que resultar em uma menor vazão perdida pela sonda. Caso nenhuma troca proporcione melhora, a composição original da sonda é mantida. Isso se repete para todos os poços presente na rota associada a uma sonda. Esse processo é repetido para todas as sondas.

O pseudo-código da Vizinhança 1 descrito na figura 4.4, mostra que cada poço da sonda (linha 3) percorre todos os seus r vizinhos mais próximos (valor passado como parâmetro) que estão na estrutura *r-vizinhos*, formada antes da execução da vizinhança (*V1*). Na linha 4 é estabelecido que o índice *poço_ atual* equivale ao índice que possui o poço x da sonda original. A sonda original é percorrida para que com as trocas de poços da solução base não se deixe de percorrer a vizinhança de algum poço e nem que esta seja percorrida mais de uma vez. Na linha 7, uma variável auxiliar *aux* recebe a sonda com os dois poços trocados de lugar, o *poço_ atual* e o vizinho j do *poço_ atual*. Sempre guardando a melhor troca dentre todas as trocas de *poço_ atual* e seus vizinhos.

Após percorrer todos os vizinhos de um poço, a *sonda_base* é atualizada com a melhor troca desse poço. Note que se nenhuma melhora foi conseguida, a sonda passada como parâmetro é a sonda retornada pelo procedimento V1, já que no início do algoritmo esta sonda é salva na variável *sonda_melhor* e se não houver melhora, esta variável não é atualizada.

```
procedure Vizinhança1_BL1 (sonda_k, r)
1.  sonda_base = Copia(sonda_k);
2.  sonda_melhor = Copia(sonda_k);
3.  Para x variando de 1 até num_poços de sonda_k  faça
4.      poço_atual = sonda_k[x];
5.      Para j variando de 1 até w  faça
6.          Se w-vizinhos[poço_atual][j] está na sonda_k então
7.              aux = troca_poços(sonda_base,poço_atual, w-vizinhos[poço_atual][j]);
8.              Se aux é melhor do que sonda_melhor então
9.                  sonda_melhor = aux;
10.             Fim Se;
11.         Fim Se;
12.     Fim Para;
13.     sonda_base = sonda_melhor;
14. Fim Para;
15. retorna sonda_melhor;
Fim Vizinhança1_BL1
```

Figura 4.4: Pseudo-código da Primeira Vizinhança da Busca Local 1

A segunda vizinhança, descrita na figura 4.5 é similar à primeira, porém neste caso, cada poço p_i é permutado com os poços que além de estarem dentre seus *r-vizinhos* mais próximos também pertencem à lista de visitas de alguma sonda que seja diferente da sonda a qual p_i pertence. Note que na linha 5 do algoritmo, a variável *poço_atual* está sendo definida como o poço escalonado para a sonda i na posição x da solução passada para o procedimento *Vizinhança2_BL1* pela mesma razão do algoritmo da vizinhança *V1*, ou seja, para que com as trocas de poços da solução base não se deixe de percorrer a vizinhança de algum poço e nem que esta seja percorrida mais de uma vez.

4.5 Busca Local BL2

Considere novamente uma solução inicial composta de k rotas (sondas) e seja um poço p_i escalonado para uma sonda s_w. A busca local 2 (*BL2*) funciona da seguinte maneira:

```
procedure Vizinhança2_BL1 (solução, r)
1.   solução_base = Copia(solução);
2.   solução_melhor = Copia(solução);
3.   Para i variando de 1 até num_sondas faça
4.       Para x variando de 1 até num_poços de sonda[i] de solução faça
5.           poço_atual = solução[i][x];
6.           Para j variando de 1 até w  faça
7.               Se w-vizinhos[poço_atual][j] não está na sonda[i] então
8.                   aux = troca_poços(solução_base,poço_atual, w-vizinhos[poço_atual][j]);
9                    Se aux é melhor do que solução_melhor então
10                       solução_melhor = aux;
11                   Fim Se;
12               Fim Se;
13           Fim Para;
14           solução_base = solução_melhor;
15       Fim Para;
16   Fim Para;
17.  retorna solução_melhor;
Fim Vizinhança2_BL1
```

Figura 4.5: Pseudo-código da Segunda Vizinhança da Busca Local 1

Um poço p_i tem sua alocação testada em todas as posições possíveis das outras $(k-1)$ sondas (sondas diferentes de s_w); a alocação que obtiver a maior economia (maior redução da vazão perdida) em relação à solução corrente será a alocação implementada.

Se a solução encontrada pela busca local não proporcionar melhora na solução corrente, então a solução corrente é mantida. Esta análise é feita para cada poço de todas as sondas.

Este algoritmo está melhor detalhado na figura 4.6, onde nas linhas 3 e 4 ele estipula a sonda e o poço a serem considerados, respectivamente, com relação à solução passada como parâmetro pelos mesmos motivos explicados na Busca Local 1 (BL1), para que com a solução alterada não se deixe de percorrer a vizinhança de algum poço e nem que esta seja percorrida mais de uma vez.

Como o poço só é testado nas posições de outras sondas diferentes da que ele está alocado originalmente, então na linha 7 é comparado se a sonda a ser testada é a mesma de origem do poço, em caso negativo testamos o poço em todas as posições da sonda. Note que na linha 8 testamos o poço nas posições 1 até *n+1* pois se o poço está entrando nesta sonda, então teremos uma posição a mais na sonda (*solução_ base[i]*). Na linha 9

a variável *aux* recebe a *sonda_ base* alterada, ou seja, com o poço *atual* na posição *w* da sonda. Depois de percorrer todas as posições de todas as sondas as quais o poço não estava alocado na solução original (passada como parâmetro), então, a melhor troca para esse poço é mantida, e para o próximo poço é feito o mesmo procedimento em cima da solução alterada (em caso de melhora). Isso é executado até que todos os poços tenham sua vizinhança percorrida.

```
procedure BL_2 ( solução )
1.   solução_base = Copia(solução);
2.   solução_melhor = Copia(solução);
3.   Para i variando de 1 até num_sondas faça
4.       Para x variando de 1 até num_poços de sonda[i] de solução faça
5.           atual = solução[i][x];
6.           Para j variando de 1 até num_sondas  faça
7.               Se j é diferente de i então
8.                   Para w variando de 1 até (num_poços+1) de solução_base[j] faça
9.                       aux = coloca_poço_na_posição(solução_base,atual, w);
10.                      Se aux é melhor do que solução_melhor então
11.                          solução_melhor = aux;
12.                      Fim Se;
13.                  Fim Para;
14.              Fim Se;
15.          Fim Para;
16.          solução_base = solução_melhor;
17.      Fim Para;
18.  Fim Para;
19.  retorna solução_melhor;
Fim BL_2
```

Figura 4.6: Pseudo-código da Busca Local 2

4.6 Busca Local BL3

Essa busca também possui duas estruturas de vizinhança, *V1* e *V2*, que são executadas uma após a outra, em *loop*, até que não haja melhora na segunda vizinhança. Somente a melhora da segunda vizinhança é considerada para que a busca local prossiga; pois, executamos alguns testes para esta busca e observamos que se a segunda vizinhança não produzir melhora, a iteração seguinte da busca local também não produzirá, independente do resultado da primeira vizinhança ter sido melhor ou não.

O pseudo-código da BL3 está descrito na figura 4.7, onde da linha 1 à linha 6 é feito um *loop* que tem como critério de parada a não melhoria da solução pela estrutura de vizinhança *V2* da BL3. Da linha 2 à 4 é feito um outro *loop* para executar a vizinhança *V2* para todas as sondas, estrutura executada na linha 3.

```
procedure BL_3 (solução_base)
1.   Faça
2.        Para i variando de 1 até num_sondas faça
3.             solução_base = Vizinhança1( i, solução_base );
4.        Fim Para;
5.        solução_base = Vizinhança2 ( solução_base, melhorou );
6.   Enquanto melhorou > 0;
7.  retorna solução_base;
 Fim BL_3
```

Figura 4.7: Pseudo-código da Busca Local 3

A primeira estrutura de vizinhança *V1* é semelhante à da busca local *BL2*, porém ao invés de verificar a melhor posição de cada poço p_i nas sondas diferentes de s_k, sonda onde p_i se encontra alocado, os testes são feitos nas posições da própria sonda s_k em que p_i está alocado, considerando a ordenação atual da sonda, e o aloca na melhor posição possível.

Um pseudo-código da vizinhança *V1* é descrito na figura 4.8. Na linha 4, a chave do poço *atual* é guardada na variável *atual* considerando a sonda original, pois se a sonda modificada fosse considerada, a vizinhança de algum poço poderia não ser visitada, ou ainda, ser visitada repetidas vezes, como já explicado anteriormente em outros algoritmos que usaram essa mesma idéia. Na linha 6 a variável *aux* recebe a *sonda_ base* alterada, com o poço *atual* na posição *w*.

Depois de executar a primeira vizinhança *V1* para todos os poços de todas as sondas, é chamada a segunda vizinhança da busca local *V2*, que nada mais é do que a Busca Local 2.

Como dito anteriormente, a razão pela qual a busca local termina, considerando a ausência de melhoria apenas no resultado da segunda vizinhança, é porque o resultado da arrumação da primeira vizinhança *V1* dará sempre o mesmo resultado, caso nenhum poço seja trocado de sonda; e quando a segunda vizinhança *V2* não atualizar a solução

```
procedure Vizinhança1_BL3 ( sonda )
1.  sonda_base = Copia(sonda);
2.  sonda_melhor = Copia(sonda);
3.  Para x variando de 1 até num_poços de sonda faça
4.      atual = sonda[x];
5.      Para w variando de 1 até num_poços de sonda  faça
6.          aux = coloca_poço_na_posição(sonda_base, atual, w);
7.          Se aux é melhor do que solução_melhor então
8.              sonda_melhor = aux;
9.          Fim Se;
10.     Fim Para;
11.     sonda_base = sonda_melhor;
12.  Fim Para;
13.  retorna sonda_melhor;
Fim Vizinhança1_BL3
```

Figura 4.8: Pseudo-código da Vizinhança 1 da Busca Local 3

corrente, nenhum poço será trocado de sonda, pois, a alocação da sonda base é mantida.

Com os algoritmos de construção e busca local descritos propusemos 6 versões iniciais GRASP, que estão especificadas na Tabela 4.1.

GRASP	Construtivo	Busca Local
G1	C1	BL1
G2	C2	BL1
G3	C1	BL2
G4	C2	BL2
G5	C1	BL3
G6	C2	BL3

Tabela 4.1: Versões GRASP

4.7 Reconexão Por Caminhos (RC)

No nosso problema, a adaptação do módulo de reconexão por caminhos (RC) é feita da seguinte forma: selecionada uma solução base e uma solução alvo, ilustradas como exemplo na figura 4.9, e posteriormente colocamos na primeira posição de cada sonda da solução base o primeiro elemento das sondas da solução alvo, gerando assim uma solução intermediária s_i. Essa primeira modificação está ilustrada na figura 4.10.

Aplicamos então uma busca local ($BL1$ ou $BL2$ ou $BL3$) sobre esta solução interme-

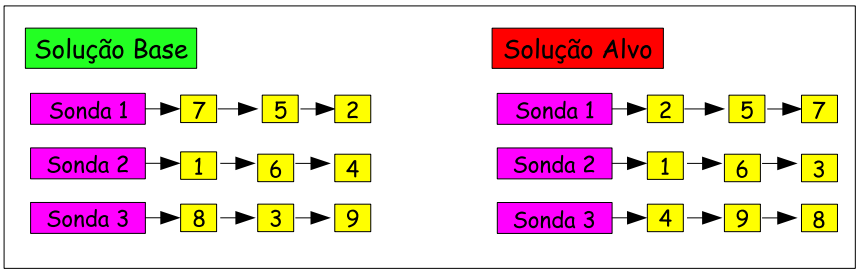

Figura 4.9: Exemplo: Soluções base e alvo selecionadas

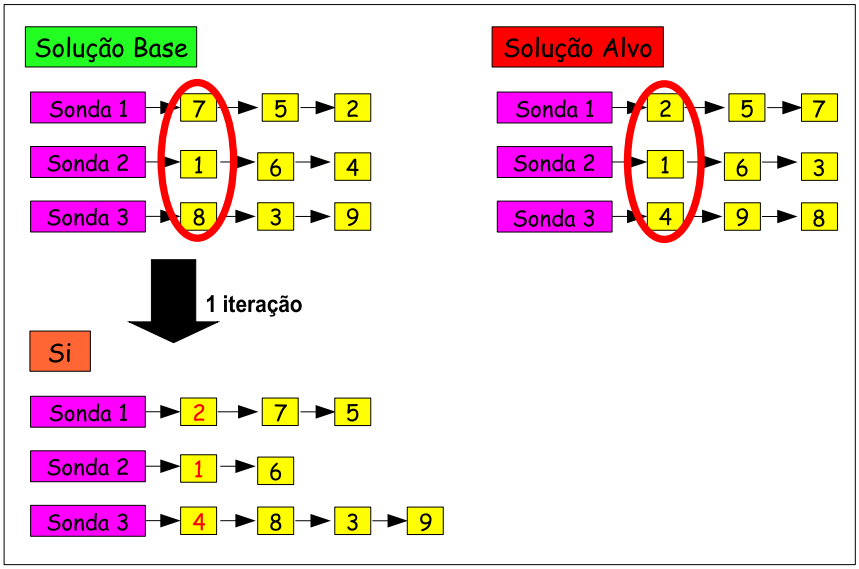

Figura 4.10: Exemplo: Primeiro passo do procedimento de RC

diária s_i e comparamos a solução refinada s_r, resultante da busca local, com a pior solução do conjunto elite (CE), atualizando este conjunto se for o caso. Exemplo ilustrado na figura 4.11.

Consideramos como a nova semente para continuar o procedimento de RC a solução intermediária s_i, ou seja, a solução gerada antes de entrar na busca local, e sobre esta seguimos as modificações. Colocamos na segunda posição de cada sonda de s_i o segundo elemento das sondas da solução alvo, e repetimos a busca local. Exemplo ilustrado na figura 4.12.

Este procedimento segue até que o último componente da solução alvo seja alocado para a atual semente s_i. Em seguida fazemos o trajeto contrário, da solução alvo para a solução base.

A *RC* pode ser aplicada à todas as soluções do CE ou para uma delas escolhida

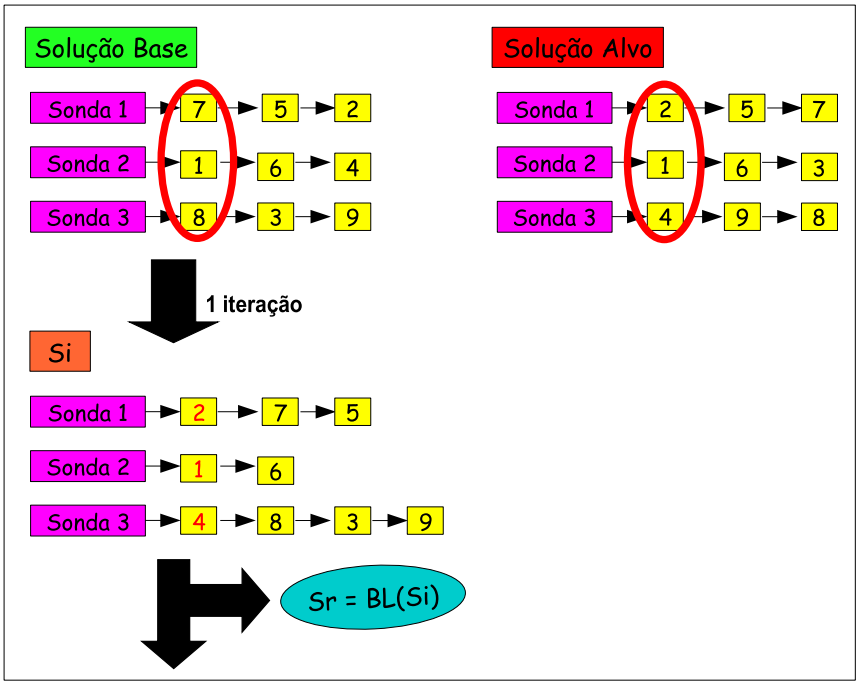

Figura 4.11: Exemplo: Segundo passo do procedimento de RC

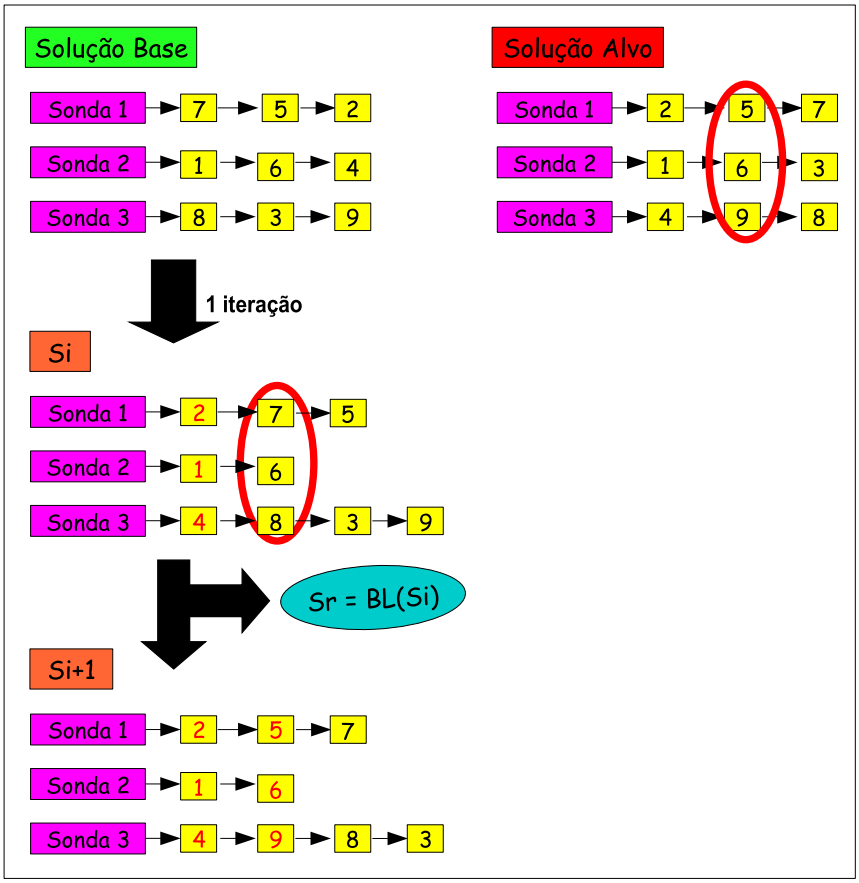

Figura 4.12: Exemplo: Terceiro passo do procedimento de RC

aleatoriamente.

Nós implementamos três versões com *RC*, que chamaremos de GRASP G7, GRASP G8 e GRASP G9. Essas duas primeiras versões são constituídas da versão GRASP G6 mais o módulo de *RC* (G6 + *RC*), diferenciadas apenas no número de vezes que o módulo de *RC* é ativado. E a versão GRASP G9 é constituída da versão GRASP G3 mais o módulo de RC com a ativação feita como no GRASP G8.

A ativação do *RC* se dá, no GRASP G7, quando a iteração do GRASP é múltipla de 50 ou quando 50% do conjunto elite for alterado desde a última ativação de *RC* (mas a primeira ativação se dá, obrigatoriamente, na iteração 50), que chamaremos de RC1. E no GRASP G8, apenas ao final da execução de todas as iterações GRASP, ou seja, apenas uma única vez, que chamaremos de RC2.

Este módulo está mais detalhado nos pseudo-códigos das figuras 4.13 e 4.14. O algoritmo da figura 4.13 mostra que o módulo *RC* é executado com uma solução base e todas as soluções do conjunto elite e que ocorre nos dois sentidos. E o pseudo-código da figura 4.14 detalha o módulo *RC* propriamente dito.

Na linha 1 do pseudo-código da figura 4.13, o conjunto elite é copiado para um conjunto auxiliar (*conj_ elite_ aux*) para que com as atualizações de *conj_ elite_ aux* (quando for o caso com as novas soluções geradas pelo módulo de RC), não se deixe de executar o módulo de RC para alguma solução do conjunto elite, e nem que este seja executado mais de uma vez.

Da linha 2 à linha 6 é feito um *loop* para fazer o módulo RC da solução corrente para todas as soluções de CE, realizados nas linhas 5 e 6 nos dois sentidos (solução alvo para a solução base e o contrário também). Este procedimento retorna o *conj_ elite_ aux* com as melhores soluções considerando tanto as soluções geradas pelo módulo de RC quanto as soluções pertencentes as CE até então.

A figura 4.14 mostra o pseudo-código do módulo de RC propriamente dito. Na linha 1 a variável *comp* guarda o índice do componente da solução intermediária que irá receber a alocação do componente de mesmo índice existente na solução alvo. Onde índice se refere à ordem (posição) em que um poço será atendido pela sonda a qual ele está alocado, ou seja, se um poço p_i está alocado na posição i da sonda s_k na solução alvo, então o poço

p_i irá ser alocado para a sonda s_k na posição i da solução intermediária.

Na linha 2 a solução intermediária, que é a solução que irá sofrer as alterações, é a princípio uma cópia da solução base. Da linha 3 à linha 16 é feito um *loop* que só termina quando todas as sondas da solução intermediária tiverem com todos os componentes iguais à solução alvo, ou seja, quando a solução intermediária estiver igual à solução alvo. Esse *loop* é controlado pela variável *finalizou*, inicializada na linha 4, e será finalizado quando a variável *finalizou* for igual ao número de sondas, pois a cada vez que uma sonda é finalizada (ou seja, está idêntica à mesma sonda da solução alvo), *finalizou* é incrementada de uma unidade, lógica apresentada nas linhas 6, 7 e 8.

Se a sonda i da solução intermediária ainda não estiver igual a mesma sonda da solução alvo, então o componente corrente (*comp*) da sonda i é igualado, linhas 9, 10 e 11. Na linha 13, após igualar o componente *comp* de todas as sondas, a busca local é executada para a solução intermediária e resulta na solução refinada, que se for o caso, será inserida no conjunto elite (linha 14).

Na linha 15 o próximo componente a ser igualado é estabelecido na variável *comp*.

```
procedure Reconexão_Geral ( solução_base, conjunto_elite )
1.    conj_elite_aux = Copia(conjunto_elite);
2.   Para i = 0 até tamanho_conjunto_elite Faça
3.          solução_alvo = conjunto_elite[i];
4.          Reconexao_Caminhos(solução_base, solução_alvo, conj_elite_aux);
5.          Reconexao_Caminhos(solução_alvo, solução_base, conj_elite_aux);
6.    Fim Faça;
Fim Reconexão_Geral
```

Figura 4.13: Pseudo-código da chamada do módulo de Reconexão por Caminhos

4.8 Versões Adaptativas

Com o intuito de obter versões GRASP capazes de se adaptarem às características particulares de cada instância, foram propostas novas versões GRASP chamadas de adaptativas. Nessas versões, a seguinte estratégia é adotada: existe uma primeira fase de treinamento, onde para cada estratégia selecionada são executadas um número fixo x de iterações GRASP. A média das qualidades das x soluções geradas são guardadas e posteriormente

```
Function Reconexao_Caminhos(solução_base,solução_alvo,conjunto_elite):solução
1.    comp = 0;
2.    solução_intermediária = Copia ( solução_base );
3.    finalizou = 0;
4.    Enquanto finalizou <> número_sondas Faça
5.          Para i = 0 até número_sondas Faça
6.                Se solução_alvo[i,comp] = NULL Faça
7.                      finalizou = finalizou + 1;
8.                Fim Faça;
9.                Senão
10.                     solução_base = Iguala_Componente(solução_alvo,comp,i);
11.               Fim Senão;
12.         Fim Para;
13.         solução_refinada = Busca_Local (solução_base);
14.         Entra_Conjunto_Elite(solução_refinada,conjunto_elite);
15.         componente = componente + 1;
16.   Fim Enquanto;
Fim Reconexao_Caminhos
```

Figura 4.14: Pseudo-código do módulo de Reconexão por Caminhos

são avaliadas e a versão com a melhor média, ou as melhores versões com as melhores médias, executam, em uma segunda fase, as iterações restantes do GRASP.

Foram implementadas ao todo doze versões, que serão chamadas de *GRASP Adaptativo* (GAdapt) mais o número da versão correspondente. Considerando 200 iterações para as heurísticas GRASP, temos as descrições das estratégias como seguem abaixo.

As estratégias são combinações dos construtivos C1 e C2 com as buscas locais BL1, BL3 e em alguns casos com ambas. A busca local BL2 não foi escolhida para compor nenhuma estratégia pois a busca BL3 já é a BL2 aprimorada.

Na primeira versão *GAdapt1* são incluídas quatro estratégias, cada uma executa 25 iterações na fase de treinamento, depois a média das soluções geradas por cada uma delas é avaliada e então a segunda melhor estratégia executa mais 30 iterações e a melhor mais 70 na segunda fase. A primeira estratégia executa C1 + BL3, a segunda, C1 + BL1 + BL3, a terceira, C2 + BL3 e a quarta, C2 + BL1 + BL3.

O pseudo-código dessa versão pode ser verificado na figura 4.8, onde da linha 1 à linha 8 é feito um *loop* de 100 repetições da execução da primeira fase do algoritmo, ou seja, a fase de treinamento. Nas primeiras 25 iterações, as soluções são geradas com a

primeira estratégia (linha 2), as 25 próximas com a segunda (linha 3), as próximas 25 com a terceira (linha 4) e as últimas 25 da fase de treinamento com a quarta (linha 5). Na linha 6 verifica-se se a solução da iteração é melhor que a *best*, e em caso positivo a *best* é atualizada. Na linha 7 calcula-se a média da qualidade das soluções geradas, da estratégia que executou na iteração i.

Da linha 9 à linha 13 é executada a segunda fase com as 100 iterações restantes do GRASP, onde a segunda melhor estratégia executa 30 iterações (linha 10) e a melhor as outra 70 (linha 11). Na linha 12 a atualização de *best* é avaliada para a segunda fase.

```
procedure GRASP_Adaptativo_V1
1.   Para i indo de 0 a 99  Faça
2.        Se (0<= i) E (i < 25)     Então solução = Estrategia_C1_BL3;
3.        Se (25<= i) E (i< 50)    Então solução = Estrategia_C1_BL1_BL3;
4.        Se (50<= i) E (i < 75)   Então solução = Estrategia_C2_BL3;
5.        Se (75<= i) E (i < 100) Então solução = Estrategia_C2_BL1_BL3;
6.        Se solução é melhor do que best  Então best = solução;
7.        Calcula_media(estrategia, i);
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.       Se (100<= i) E (i < 130) Então solução = Segunda_Melhor_Estrategia;
11.       Se (130<= i) E (i < 200) Então solução = Melhor_Estrategia;
12.       Se solução é melhor do que best  Então best = solução;
13.  Fim_Enquanto;
14.  retorna best;
Fim GRASP_Adaptativo_V1
```

Figura 4.15: Pseudo-código do GRASP Adaptativo GAdapt1

A segunda versão *GAdapt2* é muito parecida com a *GAdapt1*, diferenciando apenas nas estratégias dois e quatro, que ao invés de executar as seqüências de buscas locais BL1 + BL3, executam BL3 + BL1. Essa diferença pode ser verificada nas linhas 3 e 5 do seu pseudo-código apresentado na figura 4.16.

A terceira versão *GAdapt3* é também muito parecida com a *GAdapt1*, porém a busca local da segunda e da quarta estratégia não executam apenas uma vez a BL1 e depois uma vez a BL3, essas buscas locais são executadas uma após a outra em loop até não haver mais melhora. Assim a BL da terceira versão das estratégias dois e quatro, formam, na verdade, um VNS com essas duas buscas locais anteriores. Essa diferença está apresentada nas linhas 3 e 5 do pseudo-código apresentado na figura 4.17.

```
procedure GRASP_Adaptativo_V2
1.   Para i indo de 0 a 99  Faça
2.        Se (0<= i) E (i < 25)     Então solução = Estrategia_C1_BL3;
3.        Se (25<= i) E (i< 50)     Então solução = Estrategia_C1_BL3_BL1;
4.        Se (50<= i) E (i < 75)    Então solução = Estrategia_C2_BL3;
5.        Se (75<= i) E (i < 100) Então solução = Estrategia_C2_BL3_BL1;
6.        Se solução é melhor do que best  Então best = solução;
7.        Calcula_media(estrategia, i) ;
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.       Se (100<= i) E (i < 130) Então solução = Segunda_Melhor_Estrategia;
11.       Se (130<= i) E (i < 200) Então solução = Melhor_Estrategia;
12.       Se solução é melhor do que best  Então best = solução;
13.  Fim_Enquanto;
14.  retorna best
Fim GRASP_Adaptativo_V2
```

Figura 4.16: Pseudo-código do GRASP Adaptativo GAdapt2

```
procedure GRASP_Adaptativo_V3
1.   Para i indo de 0 a 99  Faça
2.        Se (0<= i) E (i < 25)     Então solução = Estrategia_C1_BL3;
3.        Se (25<= i) E (i< 50)     Então solução = Estrategia_C1_VNS_BL1_BL3;
4.        Se (50<= i) E (i < 75)    Então solução = Estrategia_C2_BL3;
5.        Se (75<= i) E (i < 100) Então solução = Estrategia_C2_VNS_BL1_BL3;
6.        Se solução é melhor do que best  Então best = solução;
7.        Calcula_media(estrategia, i);
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.       Se (100<= i) E (i < 130) Então solução = Segunda_Melhor_Estrategia;
11.       Se (130<= i) E (i < 200) Então solução = Melhor_Estrategia;
12.       Se solução é melhor do que best  Então best = solução;
13.  Fim_Enquanto;
14.  retorna best;
Fim GRASP_Adaptativo_V3
```

Figura 4.17: Pseudo-código do GRASP Adaptativo GAdapt3

A quarta versão *GAdapt4* é similar à *GAdapt3*, porém, a ordem de execução das buscas locais que formam um VNS são inversas, ou seja, BL3 e BL1. Diferença verificada nas linhas 3 e 5 do pseudo-código apresentado na figura 4.18.

A quinta versão *GAdapt5* está detalhada na figura 4.19. Esta versão é similar à *GAdapt1*, porém, ao invés de 25 iterações de treinamento para cada uma das 4 estratégias,

```
procedure GRASP_Adaptativo_V4
1.   Para i indo de 0 a 99  Faça
2.          Se (0<= i) E (i < 25)     Então solução = Estrategia_C1_BL3;
3.          Se (25<= i) E (i< 50)     Então solução = Estrategia_C1_VNS_BL3_BL1;
4.          Se (50<= i) E (i < 75)    Então solução = Estrategia_C2_BL3;
5.          Se (75<= i) E (i < 100) Então solução = Estrategia_C2_VNS_BL3_BL1;
6.          Se solução é melhor do que best  Então best = solução;
7.          Calcula_media(estrategia, i) ;
8.    Fim_Para ;
9.    Enquanto i < 200 Faça
10.         Se (100<= i) E (i < 130) Então solução = Segunda_Melhor_Estrategia;
11.         Se (130<= i) E (i < 200) Então solução = Melhor_Estrategia;
12.         Se solução é melhor do que best  Então best = solução;
13.   Fim_Enquanto;
14.   retorna best;
Fim GRASP_Adaptativo_V4
```

Figura 4.18: Pseudo-código do GRASP Adaptativo GAdapt4

são executadas apenas 10 iterações, o que totaliza 40 (diferença observada na linha 1 do pseudo-código). E ao invés de executar as duas melhores estratégias nas iterações restantes, apenas a melhor estratégia será executada no restante das 160 iterações (observado na linha 10).

```
procedure GRASP_Adaptativo_V5
1.   Para i indo de 0 a 39  Faça
2.          Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_BL3 ;
3.          Se (10<= i) E (i< 20)  Então solução = Estrategia_C1_BL1_BL3;
4.          Se (20<= i) E (i < 30) Então solução = Estrategia_C2_BL3;
5.          Se (30<= i) E (i < 40) Então solução = Estrategia_C2_BL1_BL3;
6.          Se solução é melhor do que best  Então best = solução;
7.          Calcula_media(estrategia, i);
8.    Fim_Para ;
9.    Enquanto i < 200 Faça
10.             solução = Melhor_Estrategia;
11.             Se solução é melhor do que best  Então best = solução;
12.   Fim_Enquanto;
13.   retorna best;
Fim GRASP_Adaptativo_V5
```

Figura 4.19: Pseudo-código do GRASP Adaptativo GAdapt5

A sexta versão *GAdapt6* é como a *GAdapt5*, alterando apenas a ordem das buscas

locais a serem executadas na segunda e quarta estratégia, ao invés de BL1 + BL3 será BL3 + BL1. Pseudo-código detalhado na figura 4.20, com as diferenças observadas nas linhas 3 e 5.

```
procedure GRASP_Adaptativo_V6
1.   Para i indo de 0 a 39  Faça
2.        Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_BL3;
3.        Se (10<= i) E (i< 20)  Então solução = Estrategia_C1_BL3_BL1;
4.        Se (20<= i) E (i < 30) Então solução = Estrategia_C2_BL3;
5.        Se (30<= i) E (i < 40) Então solução = Estrategia_C2_BL3_BL1;
6.        Se solução é melhor do que best  Então best = solução;
7.        Calcula_media(estrategia, i) ;
8.    Fim_Para ;
9.    Enquanto i < 200 Faça
10.            solução = Melhor_Estrategia;
11.            Se solução é melhor do que best  Então best = solução;
12.   Fim_Enquanto;
13.   retorna best;
Fim GRASP_Adaptativo_V6
```

Figura 4.20: Pseudo-código do GRASP Adaptativo GAdapt6

A versão *GAdapt7* é similar à *GAdapt3*, porém com 10 iterações de treinamento para cada estratégia e as 160 restantes executam a melhor estratégia. O pseudo-código de *GAdapt7* é apresentado na figura 4.21 e a diferença entre este e a versão *GAdapt3* é apresentada na linha 1 do pseudo-código mencionado.

A oitava versão *GAdapt8* é similar à *GAdapt7* alterando apenas a ordem de execução das buscas locais que formam um VNS, ou seja, ao invés de executar a ordem BL1 e posteriormente BL3 no VNS é executada a BL3 e posteriormente a BL1. O pseudo-código é apresentado na figura 4.22, e as diferenças entre o *GAdapt8* e o *GAdapt7* nas linhas 3 e 5 deste.

A nona versão *GAdapt9*, diferente das anteriores, usa apenas duas estratégias, C1 + BL3 + BL1 e C2 + BL3 + BL1, fazendo 15 iterações de treinamento para cada uma delas e o restante das iterações com a melhor das estratégias. Esta versão está mais detalhada no pseudo-código da figura 4.23, onde da linha 1 à linha 8 é feito um *loop* de 30 repetições (execução da fase de treinamento). Nas primeiras 15 iterações, as soluções são geradas com a primeira estratégia (linhas 2 e 3) e nas próximas 15 com a segunda (linha 4 e 5).

```
procedure GRASP_Adaptativo_V7
1.   Para i indo de 0 a 39  Faça
2.         Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_BL3 ;
3.         Se (10<= i) E (i< 20)  Então solução = Estrategia_C1_VNS_BL1_BL3;
4.         Se (20<= i) E (i < 30) Então solução = Estrategia_C2_BL3;
5.         Se (30<= i) E (i < 40) Então solução = Estrategia_C2_VNS_BL1_BL3;
6.         Se solução é melhor do que best  Então best = solução;
7.         Calcula_media(estrategia, i) ;
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.           solução = Melhor_Estrategia;
11.           Se solução é melhor do que best  Então best = solução;
12.  Fim_Enquanto;
13.  retorna best;
Fim GRASP_Adaptativo_V7
```

Figura 4.21: Pseudo-código do GRASP Adaptativo GAdapt7

```
procedure GRASP_Adaptativo_V8
1.   Para i indo de 0 a 39  Faça
2.         Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_BL3 ;
3.         Se (10<= i) E (i< 20)  Então solução = Estrategia_C1_VNS_BL3_BL1;
4.         Se (20<= i) E (i < 30) Então solução = Estrategia_C2_BL3;
5.         Se (30<= i) E (i < 40) Então solução = Estrategia_C2_VNS_BL3_BL1;
6.         Se solução é melhor do que best  Então best = solução;
7.         Calcula_media(estrategia, i) ;
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.           solução = Melhor_Estrategia;
11.           Se solução é melhor do que best  Então best = solução;
12.  Fim_Enquanto;
13.  retorna best;
Fim GRASP_Adaptativo_V8
```

Figura 4.22: Pseudo-código do GRASP Adaptativo GAdapt8

Na linha 6 verifica-se se a solução da iteração é melhor que a *best*, e em caso positivo a *best* é atualizada. Na linha 7 calcula-se a média da qualidade das soluções geradas, da estratégia que executou na iteração i.

Da linha 9 à linha 12 é executada a segunda fase com a melhor estratégia executando as 170 iterações restantes do GRASP (linhas 10). Na linha 11 a atualização de *best* é avaliada para a segunda fase.

```
procedure GRASP_Adaptativo_V9
1.  Para i indo de 0 a 29  Faça
2.      Se i < 15 Então
3.          solução = Estrategia_C1_BL3_BL1;
4.      Senão
5.          solução = Estrategia_C2_BL3_BL1;
6.      Se solução é melhor do que best  Então best = solução;
7.      Calcula_media(estrategia, i) ;
8.  Fim_Para ;
9.  Enquanto i < 200 Faça
10.         solução = Melhor_Estrategia;
11.         Se solução é melhor do que best  Então best = solução;
12. Fim_Enquanto;
13. retorna best;
Fim GRASP_Adaptativo_V9
```

Figura 4.23: Pseudo-código do GRASP Adaptativo GAdapt9

A décima versão *GAdapt10* é similar à *GAdapt9* só que ao invés de executar apenas uma única vez cada vizinhança da busca local, elas são executadas em loop, até não haver mais melhora, formando assim um VNS. A diferença é apresentada nas linhas 3 e 5 do pseudo-código da figura 4.24.

```
procedure GRASP_Adaptativo_V10
1.  Para i indo de 0 a 29  Faça
2.      Se i < 15 Então
3.          solução = Estrategia_C1_VNS_BL3_BL1;
4.      Senão
5.          solução = Estrategia_C2_VNS_BL3_BL1;
6.      Se solução é melhor do que best  Então best = solução;
7.      Calcula_media(estrategia, i) ;
8.  Fim_Enquanto;
9.  Enquanto i < 200 Faça
10.         solução = Melhor_Estrategia;
11.         Se solução é melhor do que best  Então best = solução;
12. Fim_Enquanto;
13. retorna best;
Fim GRASP_Adaptativo_V10
```

Figura 4.24: Pseudo-código do GRASP Adaptativo GAdapt10

A décima primeira versão *GAdapt11* faz dez iterações de treinamento para cada estratégia e o restante com a melhor das estratégias. Utiliza as seguintes estratégias: C1

+ BL3 + BL1, C3 + BL3 + BL1, C1 + BL1 + BL3 e C3 + BL1 + BL3. Essa versão está mais detalhada no seu pseudo-código da figura 4.25, onde da linha 1 à linha 8 temos a execução da fase de treinamento com 40 versões (10 para cada estratégia), e nas 10 primeiras iterações as soluções são geradas com a primeira estratégia (linha 2), nas 10 próximas com a segunda (linha 3), nas próximas 10 com a terceira (linha 4) e nas últimas 10 da fase de treinamento com a quarta (linha 5). Na linha 6 verifica-se se a solução da iteração é melhor que a *best*, e em caso positivo a *best* é atualizada. Na linha 7 calcula-se a média da qualidade das soluções geradas, da estratégia que executou na iteração i.

Da linha 9 à linha 12 é executada a segunda fase com as 160 iterações restantes do GRASP, onde somente a melhor estratégia executa (linha 10). Na linha 11 a atualização de *best* é avaliada para a segunda fase.

```
procedure GRASP_Adaptativo_V11
1.   Para i indo de 0 a 39  Faça
2.          Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_BL3_BL1;
3.          Se (10<= i) E (i< 20)  Então solução = Estrategia_C2_BL3_BL1;
4.          Se (20<= i) E (i < 30) Então solução = Estrategia_C1_BL1_BL3;
5.          Se (30<= i) E (i < 40) Então solução = Estrategia_C2_BL1_BL3;
6.          Se solução é melhor do que best  Então best = solução;
7.          Calcula_media(estrategia, i) ;
8.    Fim_Para ;
9.    Enquanto i < 200 Faça
10.             solução = Melhor_Estrategia;
11.             Se solução é melhor do que best  Então best = solução;
12.   Fim_Enquanto;
13.   retorna best;
Fim GRASP_Adaptativo_V11
```

Figura 4.25: Pseudo-código do GRASP Adaptativo GAdapt11

A décima segunda versão *GAdapt12* é muito parecida com a *GAdapt11*, porém para cada estratégia faz VNS com as buscas locais, executando repetidamente os procedimentos BL até não haver mais melhora. O pseudo-código está detalhado na figura 4.26, onde as diferenças são apresentadas nas linhas 2, 3, 4 e 5.

```
procedure GRASP_Adaptativo_V12
1.   Para i indo de 0 a 39  Faça
2.          Se (0<= i) E (i < 10)   Então solução = Estrategia_C1_VNS_BL3_BL1;
3.          Se (10<= i) E (i< 20)  Então solução = Estrategia_C2_VNS_BL3_BL1;
4.          Se (20<= i) E (i < 30) Então solução = Estrategia_C1_VNS_BL1_BL3;
5.          Se (30<= i) E (i < 40) Então solução = Estrategia_C2_VNS_BL1_BL3;
6.          Se solução é melhor do que best  Então best = solução;
7.          Calcula_media(estrategia, i) ;
8.   Fim_Para ;
9.   Enquanto i < 200 Faça
10.             solução = Melhor_Estrategia;
11.             Se solução é melhor do que best  Então best = solução;
12.  Fim_Enquanto;
13.  retorna best;
Fim GRASP_Adaptativo_V12
```

Figura 4.26: Pseudo-código do GRASP Adaptativo GAdapt12

4.9 Versões Com Filtro

Observando que a qualidade da solução originada pela fase de construção pode influenciar na qualidade da solução gerada pela fase de busca local, foi desenvolvido um módulo de filtro, aplicado em algumas versões. Neste módulo, a cada iteração GRASP, n soluções são geradas na fase de construção, e dessas n soluções apenas a melhor (*best_ contrutivo*) é guardada e passada para a fase de busca local. Como a busca local é uma busca determinística, ou seja, para uma dada solução de entrada s_x a busca local sempre gera a mesma solução de saída s_r, implementamos também uma *lista de restrições*, que guarda todas as soluções que já foram exploradas anteriormente pela busca local. Assim, o algoritmo não atualiza a solução *best_ contrutivo* sempre que a solução gerada for melhor, mas somente nos casos em que, além disso, a solução gerada também não estiver na *lista de restrição* ou *lista tabu*, pois isso significa que a solução além de ser melhor, ainda não foi explorada pela busca local.

O pseudo-código de um GRASP básico com o módulo de filtro pode ser observado na figura 4.27, onde da linha 1 à linha 11 está o *loop* das iterações GRASP que finalizam de acordo com algum critério de parada pré-estabelecido. Da linha 2 à linha 7 está o *loop* do filtro, que gera n soluções (linha 3). É verificado, para cada solução s_c gerada pelo construtivo, se a solução *best_ construtivo* será atualizada ou não; onde esta atualização só

irá ocorrer (linha 5) caso a qualidade de s_c seja melhor do que a da solução *best_ construtivo* e caso s_c não estiver na *Lista_ Restrições*, que é a lista que guarda todas as soluções que já foram analisadas pelo procedimento de busca local em alguma outra iteração GRASP (linha 4). Na linha 8 é realizado o procedimento de busca local para a *best_ construtivo*. E nas linhas 9 e 10 é feita uma atualização da solução *best* do GRASP caso a solução resultante da busca local tenha qualidade melhor.

```
procedure GRASP_Filtro
1.  Enquanto critério de parada não for satisfeito Faça
2.      Para i indo de 0 ate n-1 Faça
3.          solução = Construtivo;
4.          Se solução é melhor do que best_construtivo E
                solução não está na Lista_Restrições Então
5.              best_construtivo = solução;
6.          Fim Se;
7.      Fim Para;
8.      solução = Busca_Local (solução);
9.      Se solução é mlehor do que best Então
10.         best = solução;
11. Fim_Enquanto;
12. retorna best;
Fim GRASP_Filtro
```

Figura 4.27: Pseudo-código da versão de GRASP com filtro

Propusemos a inclusão do módulo de Filtro para algumas versões analisadas até aqui, e o teste de uma versão adaptativa com o módulo de reconexão por caminhos (RC2). Essas versões são: G6 + F, G8 + F (G6 + RC2 + F), GAdapt10 + F, GAdapt10 + RC2, GAdapt10 + RC2 + F.

Os resultados são apresentados no próximo capítulo.

Capítulo 5

Resultados Computacionais

Devido à inexistência de instâncias para o PRSM em bibliotecas públicas, geramos inicialmente dois conjuntos de problemas teste aleatoriamente, variando parâmetros tais como: o número de vértices (poços), a vazão de cada poço e a distância (tempo de percurso) entre dois poços.

Neste trabalho, para efeito de simplificação, tomamos a distância entre dois poços igual ao tempo gasto para percorrer o caminho entre eles, o horizonte de planejamento como um tempo t onde todos os poços possam ser atendidos e o número de sondas foi fixado em k = 3.

O número de vértices (poços) recebe valores iguais a: 50, 100, 200, 300, 400, 500, 700 e 1000. Foram criados dois grupos de instâncias distintas: A e B, e cada um deles possui uma instância de cada dimensão citada anteriormente. No primeiro grupo (A), a média das distâncias (tempo de percurso) entre dois poços é dominante, ou seja, é maior do que a média das vazões dos poços. A idéia, neste caso, é num processo de ordenação dos poços dar uma prioridade maior aos tempos de percurso entre dois poços, ou seja, estamos mais preocupados em gerar rotas cuja distância (tempo de percurso) seja minimizada. Neste caso, a idéia é induzir os algoritmos heurísticos a minimizar a distância percorrida e com isso, tentar minimizar a vazão total que deixou de ser coletada. Enquanto na segunda (B), ocorre o inverso. Em A a vazão recebe valores de 1 a 10 e em B a vazão recebe valores de duas a três vezes maior do que a média das distâncias entre os poços.

A distância entre os poços foram geradas aleatoriamente da seguinte forma: inicialmente os n vértices são gerados aleatoriamente no espaço R^2. De posse das coordenadas de cada vértice foi usada a métrica euclidiana para determinar a distância entre eles. Fi-

nalmente o tempo de serviço foi considerado uniforme, com valor constante igual a 1 para todos os poços.

As versões do GRASP propostas foram implementados em C, compilados com gcc e testados em um Athlon XP 2.4 GHz com 512 Mbytes de RAM. Para cada instância, cada algoritmo GRASP foi executado inicialmente três vezes utilizando os seguintes parâmetros:

- Alfa : 10% (referente ao tamanho da LRC na fase de construção).
- Critério de Parada do GRASP: Número Máximo de Iterações : 200.
- r (referente aos r vizinhos mais próximos para a Busca Local 1): 20.
- A cada execução de cada GRASP foi usada uma das sementes: 3, 7, 11.

5.1 Comparações das Versões GRASP Básicas

As tabelas apresentadas a seguir fazem a comparação da primeira etapa de testes, ou seja, são as comparações entre as versões puras de GRASP: G1, G2, G3, G4, G5 e G6.

5.1.1 Conjunto de Testes A: (Distância > Vazão)

As tabelas 5.1, 5.2, 5.3, 5.4 e 5.5 apresentam os resultados das simulações para as instâncias onde o fator *distância* domina o item *vazão*. Ou seja, estamos considerando o caso onde o custo de locomover uma sonda entre dois poços adjacentes numa rota é em média maior que o custo de manter um poço desativado para reparos.

Como as metaheurísticas, incluindo o GRASP, podem gerar soluções distintas a cada execução, mesmo utilizando os mesmos parâmetros de entrada, cada instância foi executada três vezes por cada algoritmo e a melhor solução e o desempenho médio de cada versão são ilustradas nas tabelas 5.1 e 5.2.

A tabela 5.1 apresenta os resultados considerando a melhor solução atingida por cada versão GRASP puro (de G1 a G6) para cada instância do grupo de testes A. Já a tabela

5.2 apresenta os resultados considerando a solução média para estes mesmo algoritmos e grupo de testes. A tabela 5.3 mostra o desvio da média das soluções das versões GRASP para a solução *best* de uma determinada heurística, ou seja, esse desvio é calculado em quanto por cento a solução média de uma determinada versão ficou longe da solução *best* de uma determinada instância. Este cálculo é feito pela seguinte fórmula: $(\text{solução\_média} - best) * 100/best$, onde *best* representa a melhor solução média obtida considerando todos os algoritmos analisados para uma dada instância. A tabela 5.4 mostra o melhor tempo de execução de cada algoritmo para todas as instâncias do grupo de testes A e a tabela 5.5 mostra o tempo médio das três execuções. Em todas as tabelas os valores em negrito representam os melhores resultados.

Inst.	G1	G2	G3	G4	G5	G6
50	12.538	13.264	11.769	11.781	**11.643**	11.730
100	80.921	79.235	65.044	64.625	**62.377**	63.057
200	690.055	696.650	449.148	452.788	432.981	**426.367**
300	2.072.129	2.175.974	1.232.521	1.227.870	**1.186.279**	1.203.556
400	4.830.293	4.826.058	2.470.983	2.439.908	2.413.474	**2.340.854**
500	9.279.019	8.791.048	4.335.213	4.188.671	4.254.364	**4.172.537**
700	23.313.452	23.906.789	9.669.148	9.467.649	9.354.053	**9.141.016**
1000	66.702.685	69.098.138	22.712.765	21.954.196	22.356.156	**21.691.452**

Tabela 5.1: Melhor solução de cada heurística - Conjunto de testes A

Inst.	G1	G2	G3	G4	G5	G6
50	12.986,33	13.561,33	11.835,67	12.187,33	**11.672,33**	11.763,33
100	82.342,00	80.112,67	65.636,00	65.326,33	**62.886,67**	63.590,67
200	711.417,33	707.627,33	456.054,67	454.020,33	439.508,33	**436.469,67**
300	2.099.728,67	2.206.947,00	1.237.954,00	1.235.647,00	**1.202.179,00**	1.211.853,00
400	4.967.201,33	4.891.888,00	2.518.750,00	2.459.714,33	2.436.233,33	**2.382.286,00**
500	9.427.092,67	8.944.887,67	4.374.606,00	4.222.053,67	4.287.183,33	**4.191.792,67**
700	23.923.245,67	24.162.829,67	9.733.785,33	9.512.981,00	9.448.001,33	**9.138.216,67**
1000	67.274.944,00	70.195.465,00	22.798.646,67	22.387.705,67	22.528.080,33	**21.829.369,00**

Tabela 5.2: Solução Média de cada heurística - Conjunto de testes A

Inst.	G1	G2	G3	G4	G5	G6
50	11,26%	16,18%	1,40%	4,41%	**0%**	0,78%
100	30,94%	27,39%	4,37%	3,88%	**0%**	0,78%
200	62,99%	62,13%	4,49%	4,02%	0,70%	**0%**
300	74,66%	83,58%	2,98%	2,78%	**0%**	0,80%
400	108,51%	105,34%	5,73%	3,25%	2,26%	**0%**
500	124,89%	113,39%	4,36%	0,72%	2,28%	**0%**
700	161,79%	164,42%	6,52%	4,10%	3,39%	**0%**
1000	208,19%	221,56%	4,44%	2,56%	3,20%	**0%**

Tabela 5.3: Desvio da solução média em relação à solução *best* de cada heurística - Conjunto de testes A

Inst.	G1	G2	G3	G4	G5	G6
50	0	0	0	0	0	0
100	0	0	0	0	0	0
200	0	0	100	97	165	162
300	43	22	666	664	795	809
400	246	93	1.768	1.648	2.117	2.079
500	544	465	3.567	3.515	4.466	4.258
700	1.360	1.264	10.506	9.924	12.571	12.460
1000	3.841	3.342	33.367	31.879	41.136	39.574

Tabela 5.4: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes A

Inst.	G1	G2	G3	G4	G5	G6
50	0,00	0,00	0,00	0,00	0,00	0,00
100	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	104,67	102,67	170,00	162,67
300	45,33	31,00	680,67	670,67	835,00	819,33
400	251,00	122,33	1.787,00	1.698,00	2.140,33	2.089,00
500	553,33	470,67	3.630,67	3.556,00	4.531,33	4.342,00
700	1.385,00	1.284,00	10.683,00	10.184,33	12.685,00	12.504,00
1000	3.863,00	3.404,33	33.941,33	32.619,00	41.407,67	39.961,33

Tabela 5.5: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes A

Pelos resultados ilustrados nas tabelas 5.1 a 5.5, verificamos que em termos da qualidade da solução gerada, a versão GRASP G6 (usando o algoritmo construtivo C2 e busca local BL3) apresentou os melhores resultados seguido das versões G5, G4 e G3 que tiveram resultados similares. As versões G1 e G2 apresentaram desempenho comparativo muito fraco mostrando com isso, que a busca local BL1, é menos eficaz que os demais algoritmos de busca propostos. Esses resultados mostram que para o conjunto de testes A, as versões com a busca local BL3 foram as que mais se destacaram (G6 e G5), seguidas pelas versões que utilizam a busca local BL2 (G4 e G3) e posteriormente pelas que utilizam a BL1 (G2 e G1).

É constatado ainda que entre cada par de versões que utilizam a mesma busca local, o construtivo que mais se destacou para o conjunto de testes A foi o C2. Pois entre as versões G5 e G6 que utilizam a mesma busca local BL3, G6 se mostrou mais eficiente, ainda que por pouca diferença, utilizando o construtivo C2. O mesmo ocorreu entre as versões G4 e G3, que mostrou melhores resultados a versão G4 com o construtivo C2. Porém, no caso das versões com a busca local BL1 (G2 e G1), o oposto ocorreu, a versão G1 com o construtivo C1 se destacou sobre a versão G2.

Em relação aos tempos computacionais exigidos, os resultados foram similares G1 e G2; G3 e G4; G5 e G6; não existindo nenhuma discrepância entre cada par de algoritmos com a mesma busca local, todavia, as buscas locais com melhor desempenho são também as que consomem mais tempo de execução. Em termos globais as versões G1 e G2 foram as mais rápidas, seguidas pelas versões G3 e G4, e G5 e G6.

5.1.2 Conjunto de Testes B: (Distância < Vazão)

Um segundo conjunto de testes foi efetuado, desta vez com instâncias onde o item *vazão* domina o item *distância*. Neste caso, a tendência é obtermos rotas sem o equilíbrio do tamanho destas, priorizando o item *vazão*.

Os resultados apresentados para o conjunto de testes B estão apresentados nas tabelas de 5.6 a 5.10.

A tabela 5.6 apresenta os resultados considerando a melhor solução atingida, a tabela 5.7 apresenta os resultados considerando a solução média para estes mesmo algoritmos, a tabela 5.8 mostra o desvio da média das soluções das versões GRASP para a solução *best* de uma determinada heurística. Já as tabelas 5.9 e 5.10 mostram o melhor tempo de execução de cada algoritmo e o tempo médio das três execuções, respectivamente.

Inst.	G1	G2	G3	G4	G5	G6
50	684.930	679.742	601.010	606.665	603.034	**597.598**
100	2.148.312	2.371.971	1.776.731	1.724.658	**1.696.893**	1.714.013
200	7.871.120	8.705.534	5.564.589	5.371.630	5.438.841	**5.276.815**
300	16.330.784	17.776.469	10.329.397	10.076.224	10.211.445	**9.882.047**
400	29.728.145	30.493.820	17.039.645	16.387.405	16.559.310	**16.125.353**
500	46.957.524	49.604.246	25.110.319	24.077.732	24.131.538	**23.773.128**
700	89.035.866	92.437.520	43.119.176	41.935.159	42.203.280	**41.406.137**
1000	175.959.298	184.422.209	78.094.019	75.248.009	76.983.546	**74.190.067**

Tabela 5.6: Melhor solução de cada heurística - Conjunto de testes B

Inst.	G1	G2	G3	G4	G5	G6
50	686.276,33	702.622,00	612.005,33	607.896,33	603.262,67	**600.275,33**
100	2.185.500,67	2.445.891,00	1.779.246,67	1.736.213,67	1.720.902,67	**1.718.166,33**
200	7.918.455,00	8.785.271,33	5.651.694,33	5.412.086,67	5.472.734,33	**5.327.550,33**
300	16.596.831,67	18.174.888,00	10.478.279,00	10.156.948,33	10.260.002,67	**10.004.278,33**
400	30.145.725,00	31.457.663,33	17.183.472,33	16.539.986,67	16.683.668,33	**16.152.771,33**
500	47.786.614,00	50.399.002,67	25.354.341,00	24.405.953,33	24.310.743,33	**23.917.008,67**
700	89.879.695,00	93.070.203,00	43.395.410,67	42.180.928,00	42.407.889,00	**41.598.680,00**
1000	180.360.080,67	70.195.465,00	78.884.076,67	75.695.752,00	77.017.037,00	**74.654.812,67**

Tabela 5.7: Solução Média de cada heurística - Conjunto de testes B

Inst.	G1	G2	G3	G4	G5	G6
50	14,33%	17,05%	1,95%	1,27%	0,48%	**0%**
100	27,20%	42,35%	3,55%	1,05%	0,16%	**0%**
200	48,63%	64,90%	6,08%	1,59%	2,73%	**0%**
300	65,90%	81,67%	4,74%	1,53%	2,56%	**0%**
400	86,63%	94,75%	6,38%	2,40%	3,29%	**0%**
500	99,80%	110,72%	6,01%	2,04%	1,65%	**0%**
700	116,06%	123,73%	4,32%	1,40%	1,95%	**0%**
1000	141,59%	149,62%	5,67%	1,39%	3,16%	**0%**

Tabela 5.8: Desvio da solução média em relação à solução *best* de cada heurística - Conjunto de testes B

Inst.	G1	G2	G3	G4	G5	G6
50	0	0	0	0	0	0
100	0	0	0	0	0	0
200	0	0	86	80	110	131
300	32	21	636	600	708	708
400	258	224	1.675	1.662	2.074	2.032
500	561	463	3.539	3.467	4.389	4.269
700	1.419	1.248	11.201	10.535	13.243	12.628
1000	3.928	3.433	35.807	34.936	43.668	42.168

Tabela 5.9: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes B

Inst.	G1	G2	G3	G4	G5	G6
50	0,00	0,00	0,00	0,00	0,00	0,00
100	0,00	0,00	0,00	0,00	0,00	0,00
200	0,00	0,00	95,67	82,33	120,67	133,00
300	34,33	25,67	639,00	616,67	730,33	732,67
400	264,67	226,67	1.745,67	1.682,33	2.096,33	2.080,33
500	570,33	469,33	3.659,33	3.522,00	4.475,33	4.278,00
700	1.428,00	1.254,67	11.280,33	10.754,33	13.433,67	12.920,67
1000	4.073,00	3.451,67	35.882,00	35.207,67	44.284,67	42.623,67

Tabela 5.10: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes B

Pelos resultados ilustrados nas tabelas 5.6 a 5.10, verificamos que mais uma vez, em termos da qualidade da solução gerada, a versão GRASP G6 apresentou os melhores resultados. O desempenho das demais versões, por sua vez, não mantiveram a mesma ordem do conjunto de testes A, já que neste caso, a ordem foi: G4, G5 e G3, que tiveram resultados similares. As versões G2 e G1, mais uma vez, apresentaram desempenho comparativo muito fraco.

Neste caso, houve uma competitividade maior entre as versões G6 e G4, principalmente para instâncias maiores. Isso nos mostra, que metaheurísticas são (embora menos que as heurísticas tradicionais) ainda sensíveis aos parâmetros de entrada do problema.

Mas de uma forma geral, percebemos que não ocorreu nenhuma anomalia nesta bateria de testes quando comparada com a bateria anterior. Ou seja, a versão G6 foi novamente a melhor e as de pior desempenho foram as versões G1 e G2. Uma ordenação do desempenho médio das seis versões é mostrado na tabela 5.11.

Esses resultados mostram que para o conjunto de testes B, as versões com o construtivo C2 foram as que mais se destacaram (G6 e G4), sendo que a busca local BL3 foi a que realmente se mostrou mais eficaz, deixando a versão que a utiliza (G6) com o melhor desempenho. Depois seguiram as versões com o construtivo C1 (G5 e G3), sendo mais uma vez a versão com a BL3 (G5) a que mais se destacou entre as duas. Posteriormente as que utilizam a BL1 (G2 e G1) que comprovou ser inferior as outras buscas locais, ainda que evidenciando que o construtivo C2 foi melhor também para este caso.

Em relação aos tempos computacionais exigidos, os resultados foram similares ao do conjunto de testes A.

A classificação das versões GRASP puro aqui propostas está apresentado a seguir na tabela 5.11, onde são considerados os dois conjuntos de testes A e B.

Classificação	**GRASP**	**Desvio**
1	G6	0,17%
2	G5	1,74%
3	G4	2,40%
4	G3	4,56%
5	G1	86,46%
6	G2	95,42%

Tabela 5.11: Classificação dos algoritmos GRASP baseado nos dois conjuntos de teste (A e B).

5.2 Comparações Com Métodos Exatos

Em uma segunda bateria de testes, foram gerados mais dois conjuntos compostos de instâncias pequenas, utilizados para encontrar a solução ótima usando métodos exatos através da formulação matemática aqui proposta. Essas instâncias variam seus parâmetros em 5 e 7 vértices (poços) e 2 e 3 sondas. Os demais parâmetro são gerados como descrito anteriormente. Estes também possuem os dois grupos de instâncias A e B.

É importante ressaltar que instâncias maiores foram geradas, com parâmetro de 10,

15, 20 e 30 poços variando o número de sondas em 2 e 3, porém os métodos exatos utilizados não conseguiram resolver esses problemas após 48 horas de execução e por isso não estão apresentadas nesse trabalho.

A seguir são apresentadas as tabelas do conjunto de testes de pequenas instâncias, contendo os resultados da comparação da solução obtida pelo GRASP G6, que foi a melhor versão das heurísticas GRASP puro aqui propostas com os resultados ótimos, obtidos através dos métodos exatos CPLEX e GLPK. Onde

- $*$: são as soluções com o *software* CPLEX
- $+$: são as soluções com o *software* GLPK

Os resultados obtidos nas pequenas instâncias através dos métodos exatos foram comparados com os obtidos pela melhor versão GRASP puro *G6*. Os resultados referentes ao conjuntos de testes A estão apresentados na tabela 5.12 e ao conjunto de testes B, na tabela 5.13. É observado que em 88% dos casos a versão GRASP G6 conseguiu alcançar o ótimo, porém com o tempo de execução muito inferior ao tempo utilizado pelo método exato utilizado. Apenas na instância do conjunto de testes B de sete poços e duas sondas que a versão GRASP *G6* não alcançou o valor ótimo, mas mesmo assim ficou apenas 6% distante do valor ótimo. Estes resultados demonstram uma boa confiabilidade e rapidez das heurísticas aqui propostas para a solução do PRSM.

Conjunto de Testes A: (Distância $>$ Vazão)

Poços(#)	**Sondas(#)**	**Valor Ótimo**	**Tempo(seg)**	**Solução G6**	**Tempo G6(seg)**
5	**2**	2.207^{+}	$1,00^{+}$	2.207	0,00
5	**3**	8.657^{+}	$1,00^{+}$	8.657	0,00
7	**2**	6.468^{+}	$50,00^{+}$	6.468	0,00
7	**3**	1.016^{*}	$4,11^{*}$	1.016	0,00

Tabela 5.12: Comparação G6 com a solução ótima

Conjunto de Testes B: (Distância $<$ Vazão)

Poços(#)	Sondas(#)	Valor Ótimo	Tempo(seg)	Solução G6	Tempo G6(seg)
5	**2**	27.840^{+}	$1,00^{*}$	27.840	0,00
5	**3**	16.283^{+}	$1,00^{+}$	16.283	0,00
7	**2**	37.972^{+}	$75,00^{+}$	40.422	0,00
7	**3**	42.913^{*}	$34,04^{*}$	42.913	0,00

Tabela 5.13: Comparação G6 com a solução ótima

5.3 Comparação das versões GRASP Puro com as de Reconexão por Caminhos

Nesta seção, incluímos o módulo de *Reconexão por Caminhos* (*RC*) na melhor versão GRASP gerada até o momento (GRASP G6) e na versão GRASP G3, que dentre as melhores versões puras é a primeira que possui os algoritmos de suas duas etapas (algoritmo de construção e de busca local) diferentes de G6.

As versões com *Reconexão por Caminhos* propostas são apresentadas na tabela 5.14, todas utilizando o tamanho do conjunto elite igual a 3.

GRASP	GRASP Básico	Reconexão por Caminhos
G7	G6	RC1
G8	G6	RC2
G9	G3	RC2

Tabela 5.14: Versões GRASP com RC

O impacto desta proposta pode ser vista, no caso do conjunto de testes *A*, nas tabelas 5.15, 5.16, 5.17, 5.18 e 5.19, e no caso do conjunto de testes *B* nas tabelas 5.20, 5.21, 5.22, 5.23 e 5.24.

Verificamos pelos resultados apresentados que a introdução deste módulo traz benefícios na qualidade da solução mesmo considerando a melhor versão dentre as seis versões básicas analisadas na seção anterior, contudo, como contra partida, existe um acréscimo significativo nos tempos computacionais exigidos pelas versões que possuem o módulo de RC (G7, G8 e G9).

Observamos ainda que apesar da versão G9 apresentar as qualidades de suas soluções distantes da melhor versão (G7), notamos uma melhora considerável da versão G9 para a versão G3, o que mostra que a inserção do módulo de RC realmente surte melhoras nas soluções obtidas pelas versões GRASP.

Conjunto de Testes A: (Distância > Vazão)

Os resultados computacionais obtidos para o conjunto de testes *A* estão apresentados nas tabela de 5.15 a 5.19.

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	11.730	11.658	**11.652**	11.769	11.724
100	63.057	**62.996**	**62.996**	65.044	65.028
200	426.367	**426.198**	426.367	449.148	443.087
300	1.203.556	**1.193.685**	1.194.773	1.232.521	1.229.770
400	2.340.854	**2.316.185**	2.339.234	2.470.983	2.464.523
500	4.172.537	**4.172.304**	**4.172.304**	4.335.213	4.335.213
700	9.141.016	**9.119.369**	9.120.890	9.669.148	9.669.148
1000	21.691.452	**21.677.423**	21.691.452	22.712.765	22.623.133

Tabela 5.15: Melhor solução das heurísticas - Conjunto de testes A

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	11.763,33	**11.676,67**	11.737,33	11.835,67	11.820,00
100	63.590,67	**63.486,00**	63.566,67	65.636,00	65.454,67
200	436.469,67	**435.459,33**	435.579,67	456.054,67	451.417,33
300	1.211.853,00	**1.201.940,00**	1.207.577,67	1.237.954,00	1.237.037,00
400	2.382.286,00	**2.370.571,33**	2.378.647,67	2.518.750,00	2.490.984,67
500	4.191.792,67	**4.186.731,00**	4.191.589,00	4.374.606,00	4.374.224,67
700	9.138.216,67	**9.137.709,67**	9.138.216,66	9.733.785,33	9.723.665,67
1000	21.829.369,00	**21.807.476,33**	21.829.369,00	22.798.646,67	22.759.291,33

Tabela 5.16: Solução Média de cada heurística - Conjunto de testes A

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	0,74%	**0%**	0,52%	1,36%	1,22%
100	0,16%	**0%**	0,13%	3,39%	3,10%
200	0,23%	**0%**	0,004%	4,73%	3,67%
300	0,82%	**0%**	0,47%	3,00%	2,92%
400	0,49%	**0%**	0,34%	6,25%	5,08%
500	0,12%	**0%**	0,12%	4,49%	4,48%
700	**0%**	**0%**	**0%**	6,52%	6,41%
1000	0,10%	**0%**	0,10%	4,55%	4,36%

Tabela 5.17: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes A

Os resultados experimentais das instâncias do conjunto de testes *A*, ilustrados nas tabelas 5.15 a 5.17, mostram que a inclusão do módulo de Reconexão por Caminhos (RC) sempre melhora os resultados das suas versões puras (sem RC).

Em uma análise comparativa entre as versões G3, G6, G7, G8 e G9, observamos que em termos da qualidade da solução (melhor solução e solução média) o melhor desempenho foi obtido pela versão G7 seguido da versão G8.

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	0	0	0	0	0
100	0	76	16	0	12
200	162	1.588	386	100	362
300	809	10.248	2.647	666	2.251
400	2.079	34.613	8.377	1.768	6.949
500	4.258	69.451	20.604	3.567	17.013
700	12.460	324.647	77.294	10.506	65.185
1000	39.574	1.345.148	354.572	33.367	269.085

Tabela 5.18: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes A

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	0,00	0,00	0,00	0,00	0,00
100	0,00	77,67	17,33	0,00	12,67
200	162,67	1.803,67	468,00	104,67	375,00
300	819,33	11.243,67	2.707,00	680,67	2.285,67
400	2.089,00	37.403,33	8.619,33	1.787,00	7.376,00
500	4.342,00	91.468,33	20.757,00	3.630,67	18.061,67
700	12.504,00	361.633,33	78.772,67	10.683,00	66.806,67
1000	39.961,33	1.346.362,67	358.062,00	33.941,33	285.847,67

Tabela 5.19: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes A

O bom desempenho de G7 em relação a G8 e G9 já era esperado, já que em G7, o módulo de RC é utilizado mais vezes durante uma execução do GRASP (no G8 e G9 o módulo de RC é ativado somente ao final do GRASP).

O outro fato observado é que tanto o método de construção quanto o de busca local podem influenciar de forma significativa no desempenho do GRASP. Como observado nos testes anteriores (entre as versões G1, G2, G3, G4, G5 e G6) e nas tabelas 5.15 a 5.17, as versões G3 e G9 que usam um construtivo e uma busca local diferente de G6, G7 e G8 apresentam novamente desempenho abaixo de G6, G7 e G8. Portanto, podemos constatar que a qualidade das soluções alvo e base do módulo RC são muito importante para o desempenho deste.

A justificativa pela escolha da versão G3 para a incorporação do módulo de RC, apesar desta não ter apresentado um dos melhores resultados dentre as versões GRASP puro, se dá pelo fato de querer demonstrar que a inclusão do módulo RC pode trazer melhoras tanto para versões boas, quanto para versões não tão eficientes.

Pela tabela 5.17 observamos que nem o uso do módulo RC ao final de G3 consegue produzir resultados melhores que G6.

Em relação aos tempos computacionais exigidos, também como já era esperado, as versões com RC, e em particular o G7, apresentam os maiores tempos computacionais.

Contudo, vale advertir que o critério de parada usado nestes testes (número máximo de iterações GRASP) prejudica estas versões (que possuem módulo RC) onde o trabalho por iteração é mais dispendioso. Nas seções seguintes são apresentadas análises probabilísticas empíricas justamente para melhor avaliar o tempo exigido por cada versão, aqui proposta, para atingir determinado valor alvo.

Conjunto de Testes B: (Distância < Vazão)

As tabelas 5.20 a 5.24 apresentam resultados e tempos exigidos pelos algoritmos G6, G7, G8 e G9 para o conjunto de testes *B*.

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	597.598	**589.398**	597.598	601.610	601.010
100	**1.714.013**	**1.714.013**	**1.714.013**	1.776.731	1.772.798
200	**5.276.815**	**5.276.815**	**5.276.815**	5.564.589	5.564.589
300	**9.882.047**	**9.882.047**	**9.882.047**	10.329.397	10.329.397
400	16.125.353	16.107.208	**16.071.075**	17.039.645	17.038.525
500	23.773.128	**23.740.671**	23.761.488	25.110.319	24.981.937
700	41.406.137	**41.401.473**	41.405.551	43.119.176	43.119.176
1000	74.190.067	**74.048.256**	74.119.492	78.094.019	78.088.450

Tabela 5.20: Melhor solução de cada heurística - Conjunto de testes B

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	600.275,33	**594.361,67**	599.625,67	612.005,33	611.398,67
100	**1.718.166,33**	**1.718.166,33**	**1.718.166,33**	1.779.246,67	1.775.087,67
200	5.327.550,33	**5.299.019,67**	5.311.385,00	5.651.694,33	5.649.839,67
300	10.004.278,33	**9.990.106,00**	9.992.127,33	10.478.279,00	10.461.794,33
400	16.152.771,33	16.115.395,67	**16.103.351,33**	17.183.472,33	17.105.386,33
500	23.917.008,67	**23.902.375,67**	23.913.128,67	25.354.341,00	25.286.658,33
700	41.598.680,00	**41.587.574,33**	41.589.644,33	43.395.410,67	43.383.629,33
1000	74.654.812,67	**74.597.246,33**	74.614.223,67	78.884.076,67	78.882.015,33

Tabela 5.21: Solução Média de cada heurística - Conjunto de testes B

Os resultados das tabelas de 5.20 a 5.24 comprovam novamente a superioridade das versões que incluem o módulo de RC e em particular da versão G7 que novamente apresenta os melhores resultados. Na questão tempo computacional, os resultados são similares aos verificados nas instâncias do conjunto de testes *A*.

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	1,00%	**0%**	0,89%	2,97%	2,87%
100	**0%**	**0%**	0%	3,55%	3,31%
200	1,86%	**0%**	0,23%	6,66%	6,62%
300	0,14%	**0%**	0,02%	4,89%	4,72%
400	0,31%	0,07%	**0%**	6,63%	6,14%
500	0,06%	**0%**	0,04%	6,07%	5,79%
700	**0%**	**0%**	**0%**	4,35%	4,32%
1000	**0%**	**0%**	**0%**	5,75%	5,74%

Tabela 5.22: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes B

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	0	0	0	0	0
100	0	66	12	0	9
200	131	2.032	456	86	317
300	708	8.285	2.542	636	2.066
400	2.032	34.356	7.633	1.675	7.109
500	4.269	68.341	19.781	3.539	15.523
700	12.628	286.703	76.832	11.201	65.983
1000	42.168	1.354.845	354.572	35.807	200.123

Tabela 5.23: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes B

Inst.	G6	G7 (G6+RC1)	G8 (G6+RC2)	G3	G9 (G3+RC2)
50	0,00	0,00	0,00	0,00	0,00
100	0,00	76,67	13,00	0,00	11,00
200	133,00	2.155,00	464,67	95,67	335,67
300	732,67	10.284,00	2.601,67	639,00	2.097,33
400	2.080,33	37.599,00	7.889,00	1.745,67	7.145,67
500	4.278,00	91.826,00	20.381,00	3.659,33	16.795,33
700	12.920,67	315.797,67	82.039,33	11.280,33	66.240,67
1000	42.623,67	1.375.562,33	358.062,00	35.882,00	257.313,00

Tabela 5.24: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes B

5.4 Resultados comparativos do GRASP Adaptativo

Para os resultados comparativos entre as 12 versões GRASP adaptativas propostas e a melhor versão GRASP puro (G6), primeiramente executamos testes para as instâncias de 50, 100, 200 e 300 poços (vértices), tanto para o conjunto de testes A quanto para o conjunto de testes B.

Tais resultados podem ser observados, para o conjunto de testes A, nas tabelas de 5.25 a 5.29.

Conjunto de Testes A: (Distância > Vazão)

Inst.	50	100	200	300
G6	11.730	63.057	426.367	1.203.556
GAdapt1	11.687	**62.377**	433.618	1.203.556
GAdapt2	**11.643**	**62.377**	436.529	1.203.556
GAdapt3	11.676	**62.377**	433.618	1.203.556
GAdapt4	**11.643**	**62.377**	436.529	1.203.556
GAdapt5	11.687	63.893	434.126	1.186.279
GAdapt6	**11.643**	**62.377**	**426.352**	1.206.933
GAdapt7	11.687	63.893	434.126	1.186.279
GAdapt8	**11.643**	**62.377**	**426.352**	1.206.933
GAdapt9	**11.643**	63.057	435.971	1.186.104
GAdapt10	**11.643**	63.057	435.765	**1.186.011**
GAdapt11	11.687	63.893	432.843	1.186.104
GAdapt12	11.687	63.893	434.126	**1.186.011**

Tabela 5.25: Melhor solução de cada heurística - Conjunto de testes A

Inst.	50	100	200	300
G6	11.763,33	63.590,67	436.469,67	1.211.853,00
GAdapt1	11.701,33	63.194,00	437.213,67	1.219.232,00
GAdapt2	11.686,67	63.015,67	441.369,67	1.218.514,33
GAdapt3	11.697,67	63.187,00	437.213,67	1.219.232,00
GAdapt4	11.681,67	63.015,67	441.369,67	1.216.241,33
GAdapt5	11.695,67	64.122,00	440.118,33	1.209.952,00
GAdapt6	**11.672,33**	**62.880,67**	**434.796,00**	1.215.689,67
GAdapt7	11.695,67	64.122,00	440.118,33	1.209.952,00
GAdapt8	**11.672,33**	**62.880,67**	**434.796,00**	1.214.169,00
GAdapt9	11.696,67	63.407,67	440.138,33	1.201.795,33
GAdapt10	11.693,00	63.407,67	439.834,00	**1.201.759,67**
GAdapt11	11.695,67	64.122,00	438.245,00	1.209.549,33
GAdapt12	11.710,00	64.122,00	440.118,33	1.209.513,67

Tabela 5.26: Solução Média de cada heurística - Conjunto de testes A

Inst.	50	100	200	300
G6	0,78%	1,13%	0,38%	0,84%
GAdapt1	0,25%	0,50%	0,56%	1,45%
GAdapt2	0,12%	0,21%	1,51%	1,39%
GAdapt3	0,22%	0,49%	0,56%	1,45%
GAdapt4	0,08%	0,21%	1,51%	1,20%
GAdapt5	0,20%	1,94%	1,22%	0,68%
GAdapt6	**0%**	**0%**	**0%**	1,16%
GAdapt7	0,20%	1,94%	1,22%	0,68%
GAdapt8	**0%**	**0%**	**0%**	1,03%
GAdapt9	0,21%	0,84%	1,23%	**0%**
GAdapt10	0,18%	0,84%	1,16%	**0%**
GAdapt11	0,20%	1,97%	0,79%	0,65%
GAdapt12	0,33%	1,97%	1,22%	0,64%

Tabela 5.27: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes A

A classificação das versões baseado na solução média das instâncias do conjunto de testes A pode ser observado na tabela 5.30.

Inst.	50	100	200	300
G6	0	0	162	809
GAdapt1	0	0	187	916
GAdapt2	0	0	174	853
GAdapt3	0	0	217	1.001
GAdapt4	0	0	182	929
GAdapt5	0	0	175	858
GAdapt6	0	0	186	857
GAdapt7	0	0	181	896
GAdapt8	0	0	187	1.047
GAdapt9	0	0	194	855
GAdapt10	0	0	227	1.125
GAdapt11	0	0	191	859
GAdapt12	0	0	216	1.146

Tabela 5.28: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes A

Inst.	50	100	200	300
G6	0,00	0,00	162,67	819,33
GAdapt1	0,00	0,00	201,33	924,67
GAdapt2	0,00	0,00	180,00	867,00
GAdapt3	0,00	0,00	251,33	1.022,67
GAdapt4	0,00	0,00	197,67	990,67
GAdapt5	0,00	0,00	191,67	892,33
GAdapt6	0,00	0,00	188,33	872,33
GAdapt7	0,00	0,00	217,33	930,00
GAdapt8	0,00	0,00	208,33	1.090,67
GAdapt9	0,00	0,00	197,33	874,33
GAdapt10	0,00	0,00	234,33	1.144,33
GAdapt11	0,00	0,00	198,67	888,00
GAdapt12	0,00	0,00	246,67	1.161,00

Tabela 5.29: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes A

Classificação	GRASP	Desvio
1	GAdapt8	0,26%
2	GAdapt6	0,29%
3	GAdapt10	0,55%
4	GAdapt9	0,57%
5	GAdapt3	0,68%
6	GAdapt1	0,69%
7	GAdapt4	0,75%
8	GAdapt2	0,81%
9	GAdapt11	0,90%
10	GAdapt5	1,01%
11	GAdapt7	1,01%
12	GAdapt12	1,04%
13	G6	1,26%

Tabela 5.30: Classificação dos algoritmos Adaptativos baseado no conjunto de testes A

Podemos observar, através dos resultados apresentados na tabela 5.30, que as versões similares, diferenciando-se apenas por uma possuir a busca local com VNS e a outra não,

se classificaram, quase em sua totalidade, uma seguida da outra, com a versão com VNS à frente da versão sem VNS.

Conjunto de Testes B: (Distância < Vazão)

Os resultados para o conjunto de testes B para as 12 versões adaptativas são apresentadas nas tabelas de 5.31 a 5.35.

Inst.	50	100	200	300
G6	**597.598**	1.714.013	**5.276.815**	9.882.047
GAdapt1	602.959	**1.696.893**	**5.276.815**	10.101.548
GAdapt2	**597.598**	**1.696.893**	**5.276.815**	9.873.526
GAdapt3	602.959	**1.696.893**	**5.276.815**	10.101.548
GAdapt4	**597.598**	**1.696.893**	**5.276.815**	**9.834.893**
GAdapt5	602.959	1.714.013	5.321.899	9.882.047
GAdapt6	602.959	**1.696.893**	5.321.899	9.873.526
GAdapt7	602.959	1.714.013	5.321.899	9.882.047
GAdapt8	602.959	**1.696.893**	5.321.899	**9.834.893**
GAdapt9	602.959	**1.696.893**	5.312.293	9.873.526
GAdapt10	602.959	*1.696.893*	5.290.226	**9.834.893**
GAdapt11	602.959	1.714.013	5.312.293	9.873.526
GAdapt12	602.959	1.714.013	5.290.226	**9.834.893**

Tabela 5.31: Melhor solução de cada heurística - Conjunto de testes A

Inst.	50	100	200	300
G6	**600.275,33**	1.718.166,33	5.327.550,33	10.004.278,33
GAdapt1	603.123,33	1.714.516,33	5.332.002,00	10.140.407,33
GAdapt2	603.413,67	1.716.135,33	5.323.597,67	10.026.009,33
GAdapt3	603.028,67	1.714.516,33	5.332.002,00	10.140.407,33
GAdapt4	603.214,33	1.716.135,33	**5.313.297,00**	9.994.638,67
GAdapt5	603.237,67	1.718.670,33	5.395.497,00	10.030.181,67
GAdapt6	603.123,33	**1.709.471,00**	5.394.746,00	9.999.094,67
GAdapt7	603.237,67	1.718.670,33	5.416.932,33	10.030.181,67
GAdapt8	603.123,33	**1.709.471,00**	5.391.487,00	**9.985.086,33**
GAdapt9	603.237,67	**1.709.471,00**	5.388.903,67	9.999.094,67
GAdapt10	603.237,67	**1.709.471,00**	5.374.620,67	**9.985.086,33**
GAdapt11	603.237,67	1.718.670,33	5.388.903,67	10.023.567,33
GAdapt12	603.237,67	1.718.607,33	5.400.066,00	9.994.638,67

Tabela 5.32: Solução Média de cada heurística - Conjunto de testes A

A classificação das versões baseado na solução média das instâncias do conjunto de testes B pode ser observado na tabela 5.36 e a classificação total, ou seja, com base nos resultados dos dois conjuntos de testes, A e B, na tabela 5.37.

Pela tabela 5.37 podemos ver que as versões estão muito próximas umas das outras. Escolhemos então, para uma segunda etapa de testes, 3 versões adaptativas para testar as demais instâncias (400, 500, 700 e 1000 poços) dos dois conjuntos (A e B).

Inst.	50	100	200	300
G6	**0%**	0,51%	0,27%	0,46%
GAdapt1	0,47%	0,30%	0,35%	1,56%
GAdapt2	0,52%	0,39%	0,19%	0,41%
GAdapt3	0,46%	0,30%	0,35%	1,56%
GAdapt4	0,49%	0,39%	**0%**	0,10%
GAdapt5	0,49%	0,54%	1,55%	0,45%
GAdapt6	0,47%	**0%**	1,53%	0,14%
GAdapt7	0,49%	0,54%	1,95%	0,68%
GAdapt8	0,47%	**0%**	1,47%	**0%**
GAdapt9	0,49%	**0%**	1,42%	0,14%
GAdapt10	0,49%	**0%**	1,15%	**0%**
GAdapt11	0,49%	0,54%	1,42%	0,39%
GAdapt12	0,49%	0,53%	1,63%	0,10%

Tabela 5.33: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes A

Inst.	50	100	200	300
G6	0	0	131	708
GAdapt1	0	0	144	861
GAdapt2	0	0	144	787
GAdapt3	0	0	157	937
GAdapt4	0	0	167	885
GAdapt5	0	0	133	809
GAdapt6	0	0	125	784
GAdapt7	0	0	134	840
GAdapt8	0	0	125	809
GAdapt9	0	0	160	783
GAdapt10	0	0	196	1.019
GAdapt11	0	0	168	809
GAdapt12	0	0	203	1.043

Tabela 5.34: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes A

Inst.	50	100	200	300
G6	0,00	0,00	133,00	732,67
GAdapt1	0,00	0,00	165,67	871,00
GAdapt2	0,00	0,00	149,67	806,33
GAdapt3	0,00	0,00	198,33	978,67
GAdapt4	0,00	0,00	176,67	909,33
GAdapt5	0,00	0,00	139,33	826,67
GAdapt6	0,00	0,00	141,67	810,33
GAdapt7	0,00	0,00	177,00	860,00
GAdapt8	0,00	0,00	156,00	884,33
GAdapt9	0,00	0,00	166,00	808,67
GAdapt10	0,00	0,00	204,67	1.030,00
GAdapt11	0,00	0,00	173,00	824,00
GAdapt12	0,00	0,00	223,00	1.060,33

Tabela 5.35: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes A

Classificação	GRASP	Desvio
1	GAdapt4	0,25%
2	G6	0,31%
3	GAdapt2	0,38%
4	GAdapt10	0,41%
5	GAdapt8	0,49%
6	GAdapt9	0,51%
7	GAdapt6	0,54%
8	GAdapt1	0,67%
9	GAdapt3	0,67%
10	GAdapt12	0,69%
11	GAdapt11	0,71%
12	GAdapt5	0,76%
13	GAdapt7	0,92%

Tabela 5.36: Classificação dos algoritmos Adaptativos baseado no conjunto de testes B

Classificação	GRASP	Desvio
1	GAdapt8	0,38%
2	GAdapt6	0,42%
3	GAdapt10	0,48%
4	GAdapt4	0,50%
5	GAdapt9	0,54%
6	GAdapt2	0,60%
7	GAdapt3	0,67%
8	GAdapt1	0,68%
9	G6	0,79%
10	GAdapt11	0,81%
11	GAdapt12	0,87%
12	GAdapt5	0,89%
13	GAdapt7	0,97%

Tabela 5.37: Classificação dos algoritmos Adaptativos baseado nos dois conjuntos de testes (A e B).

As versões escolhidas foram: GAdapt4, GAdapt10 e GAdapt12. Essas escolhas se justificam da seguinte maneira: O GAdapt10 foi a terceira melhor versão na classificação total dentre as versões adaptativas e o GAdapt4 a quarta melhor. Não optamos pelas versões GAdapt8 e GAdapt6, melhor e segunda melhor respectivamente na classificação total, pois estas versões se mostraram muito instáveis, com desempenho dependente das características da instância, já que ficaram muito bem classificadas' para o conjunto de testes A, mas se mostraram medianas nos testes com o conjunto B.

Apesar de a versão GAdapt4 ter demonstrado a mesma instabilidade, tendo um desempenho muito bom no conjunto de testes B, porém mediano no conjunto de testes A, esta foi a única versão capaz de superar a versão pura G6 no conjunto de testes B, por isso a selecionamos. Ja a versão GAdapt10 se mostrou eficiente e estável, se mantendo entre as 4 primeiras posições nos dois conjuntos de testes.

A versão GAdapt12 foi escolhida, apesar de o baixo desempenho comparado as outras versões adaptativas, pois esta apresentou os melhores resultados se tratando da maior instância de testes (300 poços) como podemos observar nas tabela 5.25 e 5.31. E como o intuito é testar agora as maiores instâncias, optamos por selecionar a versão GAdapt12.

Conjunto de Testes A: (Distância > Vazão)

A seguir são apresentados os resultados da comparação das três versões adaptativas selecionadas (GAdapt4, GAdapt10 e GAdapt12) com a melhor versão GRASP puro (G6) para as instâncias de 400 a 1000 poços (vértices) do conjunto de testes A.

Inst.	400	500	700	1000
G6	**2.340.854**	4.172.537	9.141.016	21.691.452
GAdapt4	**2.340.854**	4.175.450	9.120.890	**21.678.695**
GAdapt10	**2.340.854**	**4.172.148**	**9.112.446**	**21.678.695**
GAdapt12	**2.340.854**	**4.172.148**	**9.112.446**	**21.678.695**

Tabela 5.38: Melhor solução de cada heurística - Conjunto de testes A

Inst.	400	500	700	1000
G6	2.382.286,00	**4.191.792,67**	9.138.216,67	**21.829.369,00**
GAdapt4	2.384.632,00	4.210.646,33	9.173.382,67	21.953.094,67
GAdapt10	**2.378.685,00**	4.213.928,00	**9.130.825,67**	21.939.985,33
GAdapt12	**2.378.685,00**	4.203.849,00	9.242.570,33	21.939.985,33

Tabela 5.39: Solução Média de cada heurística - Conjunto de testes A

Inst.	400	500	700	1000
G6	0,15%	**0%**	0,08%	**0%**
GAdapt4	0,25%	0,45%	0,47%	0,57%
GAdapt10	**0%**	0,53%	**0%**	0,51%
GAdapt12	**0%**	0,29%	1,22%	0,51%

Tabela 5.40: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes A

Inst.	400	500	700	1000
G6	2.079	4.258	12.460	39.574
GAdapt4	2.567	5.115	15.161	48.214
GAdapt10	2.786	5.941	17.201	56.534
GAdapt12	2.892	6.142	18.573	58.569

Tabela 5.41: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes A

A classificação das versões baseado na solução média das instâncias do conjunto de testes A pode ser observado na tabela 5.43.

Conjunto de Testes B: (Distância < Vazão)

Inst.	400	500	700	1000
G6	2.089,00	4.342,00	12.504,00	39.961,33
GAdapt4	2.621,33	5.403,67	15.589,33	50.675,67
GAdapt10	2.844,67	6.064,00	17.740,00	57.125,33
GAdapt12	2.921,00	6.569,67	19.403,33	59.160,00

Tabela 5.42: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes A

Classificação	GRASP	Desvio
1	G6	0,06%
2	GAdapt10	0,26%
3	GAdapt4	0,44%
4	GAdapt12	0,51%

Tabela 5.43: Classificação dos algoritmos GRASP baseado nos dois conjuntos de teste A

A seguir são apresentados, nas tabelas de 5.44 à 5.48, os resultados da comparação das três versões adaptativas selecionadas (GAdapt4, GAdapt10 e GAdapt12) com a melhor versão GRASP puro (G6) para as instâncias de 400 a 1000 poços (vértices) do conjunto de testes B.

Inst.	400	500	700	1000
G6	16.125.353	23.773.128	41.406.137	**74.190.067**
GAdapt4	**16.106.413**	23.816.385	41.593.056	74.421.517
GAdapt10	**16.106.413**	**23.744.021**	**41.218.103**	74.421.517
GAdapt12	**16.106.413**	**23.744.021**	**41.218.103**	74.421.517

Tabela 5.44: Melhor solução de cada heurística - Conjunto de testes B

Inst.	400	500	700	1000
G6	16.152.771,33	23.917.008,67	41.598.680,00	74.654.812,67
GAdapt4	16.139.247,33	23.950.914,00	41.666.646,00	**74.634.682,67**
GAdapt10	**16.138.867,67**	**23.869.224,67**	**41.458.979,67**	**74.634.682,67**
GAdapt12	**16.138.867,67**	**23.869.224,67**	41.512.537,67	**74.634.682,67**

Tabela 5.45: Solução Média de cada heurística - Conjunto de testes B

Inst.	400	500	700	1000
G6	0,09%	0,20%	0,34%	0,03%
GAdapt4	0,002%	0,34%	0,50%	**0%**
GAdapt10	**0%**	**0%**	**0%**	**0%**
GAdapt12	**0%**	**0%**	0,13%	**0%**

Tabela 5.46: Desvio da solução média em relação a solução *best* de cada heurística - Conjunto de testes B

A classificação das versões baseado na solução média das instâncias do conjunto de testes B pode ser observado na tabela 5.49.

A classificação total desta segunda etapa de testes, considerando os dois conjuntos de testes A e B, pode ser observado na tabela 5.50.

Inst.	400	500	700	1000
G6	2.032	4.269	12.628	42.168
GAdapt4	2.414	5.075	15.575	55.078
GAdapt10	2.765	5.704	18.085	59.730
GAdapt12	2.836	6.040	18.599	62.930

Tabela 5.47: Melhor tempo de execução de cada heurística (segundos) - Conjunto de testes B

Inst.	400	500	700	1000
G6	2.080,33	4.278,00	12.920,67	42.623,67
GAdapt4	2.518,33	5.191,67	16.762,00	55.684,00
GAdapt10	2.796,67	5.845,33	18.211,33	60.789,00
GAdapt12	2.873,33	6.110,67	19.005,33	63.308,00

Tabela 5.48: Tempo médio de execução de cada heurística (segundos) - Conjunto de testes B

Classificação	GRASP	Desvio
1	GAdapt10	0,00%
2	GAdapt12	0,03%
3	G6	0,17%
4	GAdapt4	0,21%

Tabela 5.49: Classificação dos algoritmos GRASP baseado nos dois conjuntos de teste B

Classificação	GRASP	Desvio
1	G6	0,12%
2	GAdapt10	0,13%
3	GAdapt12	0,27%
4	GAdapt4	0,33%

Tabela 5.50: Classificação Total dos algoritmos GRASP baseado nos dois conjuntos de teste (A e B).

5.5 Resultados da melhor versão GRASP Adaptativa com Reconexão por Caminhos

Após obter a melhor versão adaptativa (GAdapt10), incluímos nesta o módulo de Reconexão por Caminhos *RC*2 e comparamos os resultados com a versão G8 (G6 + RC2).

Conjunto de Testes A: (Distância $>$ Vazão)

A seguir são apresentados os resultados da comparação da versão *GAdapt10+RC2* e *G8* apresentadas nas tabelas de 5.51 a 5.52, referentes ao conjunto de testes A.

Conjunto de Testes B: (Distância $<$ Vazão)

A seguir são apresentados os resultados da comparação da versão *GAdapt10+RC2* e *G8* apresentadas nas tabelas de 5.53 a 5.54, referentes ao conjunto de testes B.

Inst.	Melhor Solução		Solução Média	
	G8	**GAdapt10+RC2**	**G8**	**GAdapt10+RC2**
50	**11.652**	11.730	**11.737,33**	11.748,00
100	**62.996**	**62.996**	**63.566,67**	**63.566,67**
200	426.367	**426.352**	435.579,67	**434.558,00**
300	1.194.773	**1.194.756**	1.207.577,67	**1.203.640,00**
400	**2.339.234**	**2.339.234**	2.378.647,67	**2.377.292,67**
500	4.172.304	**4.172.148**	4.191.589,00	**4.188.896,33**
700	9.120.890	**9.112.446**	9.138.216,66	**9.120.468,00**
1000	**21.691.452**	**21.658.695**	21.829.369,00	**21.795.946,67**

Tabela 5.51: Melhor solução e solução média de cada heurística - Conjunto de testes A

Inst.	Melhor Tempo		Tempo Médio	
	G8	**GAdapt10+RC2**	**G8**	**GAdapt10+RC2**
50	0	0	0,00	0,00
100	16	24	17,33	24,67
200	386	537	468,00	653,33
300	2.647	3.649	2.707,00	3.755,00
400	8.377	11.541	8.619,33	11.743,67
500	20.604	28.083	20.757,00	29.253,00
700	77.294	114.707	78.772,67	117.644,33
1000	354.572	479.026	358.062,00	508.104,67

Tabela 5.52: Melhor tempo e tempo médio de cada heurística - Conjunto de testes A

Inst.	Melhor Solução		Solução Média	
	G8	**GAdapt10+RC2**	**G8**	**GAdapt10+RC2**
50	**597.598**	**597.598**	**599.625,67**	**599.625,67**
100	1.714.013	**1.696.893**	1.718.166,33	**1.709.471,00**
200	5.276.815	**5.253.958**	5.311.385,00	**5.304.154,67**
300	9.882.047	**9.834.893**	9.992.127,33	**9.960.804,00**
400	16.071.075	**16.002.710**	16.103.351,33	**16.067.700,00**
500	23.761.488	**23.707.529**	23.913.128,67	**23.849.365,33**
700	41.405.551	**41.217.517**	41.589.644,33	**41.373.074,00**
1000	74.119.492	**74.012.968**	74.614.223,67	**74.584.324,33**

Tabela 5.53: Melhor solução e solução média de cada heurística - Conjunto de testes B

Inst.	Melhor Tempo		Tempo Médio	
	G8	**GAdapt10+RC2**	**G8**	**GAdapt10+RC2**
50	0	0	0,00	0,00
100	12	17	13,00	19,00
200	456	659	464,67	669,33
300	2.542	3.291	2.601,67	3.541,67
400	7.633	10.567	7.889,00	11.002,67
500	19.781	26.922	20.381,00	27.759,33
700	76.832	111.896	82.039,33	118.133,33
1000	354.572	521.597	358.062,00	541,785,67

Tabela 5.54: Melhor tempo e tempo médio de cada heurística - Conjunto de testes B

A classificação das versões baseado na solução média das instâncias do conjunto de testes A e B pode ser observado na tabela 5.55.

Classificação	GRASP	Desvio
1	GAdapt10 + RC2	0,00%
2	G8 (G6 + RC2)	0,18%

Tabela 5.55: Classificação dos algoritmos GRASP G6 e GAdapt10 com RC.

Os resultados empíricos observados mostram que as versões adaptativas com reconexão por caminhos conseguem, na média, superar os resultados de G8.

Mesmo sendo resultados empíricos, o que se buscava nesta proposta era obter versões que tivessem um comportamento mais regular diante de diferentes tipos de instâncias.

Os resultados mostraram que, em parte, isso foi obtido, mas devemos ter conciência de que qualquer afirmação sobre a supremacia ou não desta versão necessitaria de mais estudos.

5.6 Resultados comparativos do GRASP com Filtro

Nesta seção, selecionamos algumas versões já descritas e testadas anteriormente e incluímos o módulo de *Filtro*. As versões selecionadas, como já dito anteriormente, foram G6, G8 (G6 + RC2), GAdapt10 e GAdapt10 + RC2, e as novas versões (com filtro) passaram a se chamar G6 + F, G8 + F, GAdapt10 + F e GAdapt10 + RC2 + F, respectivamente.

Os resultados da comparação dessas versões com as versões sem o filtro estão apresentados a seguir. Os resultados para o conjunto de testes A estão representados nas tabelas de 5.56 à 5.63 e os referentes ao conjunto de testes B nas tabelas de 5.64 à 5.71.

Inst.	50	100	200	300
G6	11.730	63.057	426.367	1.203.556
G6 + F	11.694	62.968	426.367	1.193.269
G8	11.652	62.996	426.367	1.194.773
G8 + F	**11.632**	**62.766**	**426.198**	1.193.269
GAdapt10	11.643	63.057	435.765	**1.186.011**
GAdapt10 + F	11.686	62.968	430.291	1.204.212
GAdapt10 + RC2	11.730	62.996	426.352	1.194.756
GAdapt10 + RC2 + F	11.659	**62.766**	429.280	1.204.117

Tabela 5.56: Melhor solução de cada heurística - Conjunto de testes A - De 50 à 300 poços

Os resultados introduzindo o filtro mostram que estes tendem a melhorar o desempenho do GRASP associado em termos da qualidade da solução gerada sem onerar significativamente os tempos computacionais exigidos.

Inst.	400	500	700	1000
G6	2.340.854	4.172.537	9.141.016	21.691.452
G6 + F	2.340.854	4.116.834	9.150.761	21.719.970
G8	**2.339.234**	4.172.304	9.120.890	21.691.452
G8 + F	2.340.854	**4.106.737**	9.150.761	21.719.970
GAdapt10	2.340.854	4.172.148	9.112.446	21.678.695
GAdapt10 + F	2.364.113	4.122.498	9.149.124	21.679.266
GAdapt10 + RC2	**2.339.234**	4.172.148	9.112.446	21.658.695
GAdapt10 + RC2 + F	2.364.113	4.122.498	**9.103.759**	**21.644.495**

Tabela 5.57: Melhor solução de cada heurística - Conjunto de testes A - De 400 à 1000 poços

Inst.	50	100	200	300
G6	11.763,33	63.590,67	436.469,67	1.211.853,00
G6 + F	11.713,33	63.232,00	430.794,00	**1.200.666,67**
G8	11.737,33	63.566,67	435.579,67	1.207.577,67
G8 + F	**11.666,33**	**63.144,67**	**429.080,67**	**1.200.666,67**
GAdapt10	11.693,00	63.407,67	439.834,00	1.201.759,67
GAdapt10 + F	11.693,67	63.295,67	435.572,67	1.210.508,67
GAdapt10 + RC2	11.748,00	63.566,67	434.558,00	1.203.640,00
GAdapt10 + RC2 + F	11.684,67	63.228,33	435.235,67	1.209.577,67

Tabela 5.58: Solução média de cada heurística - Conjunto de testes A - De 50 à 300 poços

Inst.	400	500	700	1000
G6	2.382.286,00	4.191.792,67	9.138.216,67	21.829.369,00
G6 + F	**2.371.289,33**	4.123.065,67	9.179.398,00	21.839.890,67
G8	2.378.647,67	4.191.589,00	9.138.216,66	21.829.369,00
G8 + F	**2.371.289,33**	**4.119.700,00**	9.178.603,33	21.822.512,67
GAdapt10	2.378.685,00	4.213.928,00	9.130.825,67	21.939.985,33
GAdapt10 + F	2.376.955,67	4.165.358,67	9.297.516,33	21.737.796,67
GAdapt10 + RC2	2.377.292,67	4.188.896,33	9.120.468,00	21.795.946,67
GAdapt10 + RC2 + F	2.376.955,67	4.165.358,67	**9.117.276,33**	**21.774.998,67**

Tabela 5.59: Solução média de cada heurística - Conjunto de testes A - De 400 à 1000 poços

Inst.	50	100	200	300
G6	0	0	162	809
G6 + F	0	0	150	842
G8	0	16	386	2.647
G8 + F	0	18	511	2.956
GAdapt10	0	0	227	1.125
GAdapt10 + F	0	0	213	1.103
GAdapt10 + RC2	0	24	537	3.649
GAdapt10 + RC2 + F	0	18	682	3.774

Tabela 5.60: Melhor tempo de cada heurística - Conjunto de testes A - De 50 à 300 poços

Em termos gerais, os resultados ilustrados até o momento indicam um caminho promissor para as técnicas heurísticas híbridas incorporando conjuntamente conceitos de filtro, Reconexão por Caminhos e versões adaptativas.

Inst.	400	500	700	1000
G6	2.079	4.258	12.460	39.574
G6 + F	2.153	4.317	12.808	41.334
G8	8.377	20.604	77.294	354.572
G8 + F	8.759	21.447	83.982	365.920
GAdapt10	2.786	5.941	17.201	56.534
GAdapt10 + F	2.717	5.914	17.764	55.395
GAdapt10 + RC2	11.541	28.083	114.707	479.026
GAdapt10 + RC2 + F	10.709	28.034	114.984	487.829

Tabela 5.61: Melhor tempo de cada heurística - Conjunto de testes A - De 400 à 1000 poços

Inst.	50	100	200	300
G6	0,00	0,00	162,67	819,33
G6 + F	0,00	0,00	156,67	848,00
G8	0,00	17,33	468,00	2.707,00
G8 + F	0,00	18,33	537,67	2.992,33
GAdapt10	0,00	0,00	234,33	1.144,33
GAdapt10 + F	0,00	0,00	430.291	1.202,33
GAdapt10 + RC2	0,00	17,33	468,00	2.707,00
GAdapt10 + RC2 + F	0,00	21,00	691,00	3.996,33

Tabela 5.62: Tempo médio de cada heurística - Conjunto de testes A - De 50 à 300 poços

Inst.	400	500	700	1000
G6	2.089,00	4.342,00	12.504,00	39.961,33
G6 + F	2.174,67	4.379,00	12.922,67	41.405,00
G8	8.619,33	20.757,00	78.772,67	358.062,00
G8 + F	8.911,33	22.270,33	86.093,33	387.246,00
GAdapt10	2.844,67	6.064,00	17.740,00	57.125,33
GAdapt10 + F	2.786,00	6.195,67	18.195,67	56.161,67
GAdapt10 + RC2	8.619,33	20.757,00	117.644,33	508.104,67
GAdapt10 + RC2 + F	11.268,67	29.656,67	119.024,67	493.871,33

Tabela 5.63: Tempo médio de cada heurística - Conjunto de testes A - De 400 à 1000 poços

Inst.	50	100	200	300
G6	597.598	1.714.013	5.276.815	9.882.047
G6 + F	**589.398**	1.687.602	**5.183.283**	10.040.547
G8	597.598	1.714.013	5.276.815	9.882.047
G8 + F	**589.398**	**1.682.387**	**5.183.283**	10.040.547
GAdapt10	602.959	1.696.893	5.290.226	**9.834.893**
GAdapt10 + F	**589.398**	1.687.602	**5.183.283**	10.022.777
GAdapt10 + RC2	597.598	1.696.893	5.253.958	**9.834.893**
GAdapt10 + RC2 + F	**589.398**	**1.682.387**	**5.183.283**	10.022.777

Tabela 5.64: Melhor solução de cada heurística - Conjunto de testes B - De 50 à 300 poços

5.7 Análise Probabilística dos algoritmos GRASP

Uma das limitações de heurísticas tradicionais de construção e busca local, bem como de algumas metaheurísticas, é a sua grande sensibilidade com os parâmetros de entrada do

Inst.	400	500	700	1000
G6	16.125.353	23.773.128	41.406.137	74.190.067
G6 + F	16.121.032	23.300.090	41.138.961	74.653.680
G8	16.071.075	23.761.488	41.405.551	74.119.492
G8 + F	16.121.032	**23.282.325**	41.138.961	74.648.228
GAdapt10	16.106.413	23.744.021	41.218.103	74.421.517
GAdapt10 + F	16.114.300	23.501.621	41.088.758	74.157.251
GAdapt10 + RC2	**16.002.710**	23.707.529	41.217.517	74.012.968
GAdapt10 + RC2 + F	16.047.279	23.484.156	**41.078.848**	**73.995.256**

Tabela 5.65: Melhor solução de cada heurística - Conjunto de testes B - De 400 à 1000 poços

Inst.	50	100	200	300
G6	600.275,33	1.718.166,33	5.327.550,33	10.004.278,33
G6 + F	**593.779,00**	1.704.553,67	5.238.682,00	10.076.362,67
G8	599.625,67	1.718.166,33	5.311.385,00	9.992.127,33
G8 + F	**593.779,00**	**1.695.564,33**	**5.237.402,00**	10.076.085,00
GAdapt10	603.237,67	1.709.471,00	5.374.620,67	9.985.086,33
GAdapt10 + F	**593.779,00**	1.718.373,67	5.238.682,00	10.045.387,67
GAdapt10 + RC2	599.625,67	1.709.471,00	5.304.154,67	**9.960.804,00**
GAdapt10 + RC2 + F	**593.779,00**	1.707.959,67	**5.237.402,00**	10.045.156,00

Tabela 5.66: Solução média de cada heurística - Conjunto de testes B - De 50 à 300 poços

Inst.	400	500	700	1000
G6	16.152.771,33	23.917.008,67	41.598.680,00	74.654.812,67
G6 + F	16.146.111,00	23.543.148,00	41.389.911,67	74.653.680,00
G8	16.103.351,33	23.913.128,67	41.589.644,33	74.614.223,67
G8 + F	16.125.177,33	**23.525.994,67**	**41.296.407,67**	74.828.380,67
GAdapt10	16.138.867,67	23.869.224,67	41.458.979,67	74.634.682,67
GAdapt10 + F	16.126.459,00	23.736.215,67	41.567.924,33	74.511.734,67
GAdapt10 + RC2	**16.067.700,00**	23.849.365,33	41.373.074,00	74.584.324,33
GAdapt10 + RC2 + F	16.089.456,67	23.728.520,00	41.345.549,67	**74.368.496,33**

Tabela 5.67: Solução média de cada heurística - Conjunto de testes B - De 400 à 1000 poços

Inst.	50	100	200	300
G6	0	0	131	708
G6 + F	0	0	124	771
G8	0	12	456	2.542
G8 + F	0	13	436	2.680
GAdapt10	0	0	196	1.019
GAdapt10 + F	0	0	201	988
GAdapt10 + RC2	0	17	659	3.291
GAdapt10 + RC2 + F	0	15	591	3.331

Tabela 5.68: Melhor tempo de cada heurística - Conjunto de testes B - De 50 à 300 poços

problema. Ou seja, muitas vezes uma pequena mudança nos dados de entrada podem provocar mudanças drásticas no comportamento destes algoritmos.

Nesta seção mostramos o comportamento médio dos algoritmos aqui propostos. Estes

Inst.	400	500	700	1000
G6	2.032	4.269	12.628	42.168
G6 + F	2.053	4.378	13.684	42.987
G8	7.633	19.781	76.832	354.572
G8 + F	8.176	19.794	81.389	277.206
GAdapt10	2.765	5.704	18.085	59.730
GAdapt10 + F	2.650	5.956	18.285	59.438
GAdapt10 + RC2	10.567	26.922	111.896	521.597
GAdapt10 + RC2 + F	10.447	28.095	113.433	537.710

Tabela 5.69: Melhor tempo de cada heurística - Conjunto de testes B - De 400 à 1000 poços

Inst.	50	100	200	300
G6	0,00	0,00	133,00	732,67
G6 + F	0,00	0,00	131,33	795,00
G8	0,00	13,00	464,67	2.601,67
G8 + F	0,00	14,33	467,00	2.723,33
GAdapt10	0,00	0,00	204,67	1.030,00
GAdapt10 + F	0,00	0,00	203,33	1.001,00
GAdapt10 + RC2	0,00	19,00	669,33	3.541,67
GAdapt10 + RC2 + F	0,00	16,33	620,00	3.414,33

Tabela 5.70: Tempo médio de cada heurística - Conjunto de testes B - De 50 à 300 poços

Inst.	400	500	700	1000
G6	2.080,33	4.278,00	12.920,67	42.623,67
G6 + F	2.089,67	4.478,33	13.789,33	43.505,00
G8	7.889,00	20.381,00	82.039,33	358.062,00
G8 + F	8.525,33	21.241,67	87.089,67	351.841,33
GAdapt10	2.796,67	5.845,33	18.211,33	60.789,00
GAdapt10 + F	2.710,67	6.043,33	18.878,33	59.965,67
GAdapt10 + RC2	11.002,67	27.759,33	118.133,33	541,785,67
GAdapt10 + RC2 + F	10.707,67	28.810,33	129.157,33	559.430,00

Tabela 5.71: Tempo médio de cada heurística - Conjunto de testes B - De 400 à 1000 poços

mostram como as melhores versões apresentam um nível de robustez promissor. Para avaliar o desempenho, colocamos um valor alvo a ser atingido pelos algoritmos. Estes alvos representam valores sub ótimos resultantes de alguma das versões propostas neste trabalho. No nosso caso, devido ao desconhecimento do valor ótimo na maioria das instâncias avaliadas, colocamos como alvo soluções geradas pelas heurísticas propostas.

Para isso, cada versão foi executada independentemente 100 vezes para cada instância utilizando sempre os mesmos parâmetros de entrada. O critério de parada do GRASP foi modificado para quando este atingir uma solução com valor menor ou igual ao valor alvo ou atingir um tempo limite, que foi calculado em aproximadamente três vezes o maior tempo de execução considerando todas as heurísticas GRASP para a instância em questão

nos testes anteriores.

Para cada execução i, armazena-se o tempo de cpu (t_i). O resultado de cada algoritmo é então plotado associando com o *i-ésimo* menor tempo de cpu, uma probabilidade $p_i = (i - 0,5)/100$, gerando pontos no espaço R^2 da forma $z_i = (t_i, p_i)$, para $i = 1, 2, ..., 100$ [35].

Para esta análise selecionamos apenas parte dos algoritmos propostos. Em uma primeira etapa analisamos as versões GRASP puro: G1, G2, G3, G4, G5 e G6, e em uma segunda etapa analisamos os algoritmos: G6 e G7, para uma melhor avaliação do módulo RC.

A escolha destas últimas duas versões tem como objetivo mostrar o impacto do módulo RC na melhor versão pura (G6).

As comparações entre G3 e G9 ou G6 e G8 não puderam ser feitas já que em G8 e G9 o módulo RC somente é ativado após a parada do GRASP G6 e G3, respectivamente.

As figuras de 5.1 à 5.4 ilustram os resultados obtidos pelas versões G1 à G6. Note que cada curva nestes gráficos representa a convergência empírica de cada algoritmo e observe que quanto *mais à esquerda* estiver a curva, melhor (mais rápido) será a convergência do algoritmo.

A análise probabílistica foi executada em uma primeira etapa para todas as versões puras (G1, G2, G3, G4, G5 e G6), porém note que as curvas das versões G1 e G2 não aparecem nas figuras de 5.1 à 5.4, pois pelo seu desempenho pouco eficiente, estas versões não conseguiram atingir o alvo em nenhuma das 100 execuções, dentro do tempo limite estabelecido.

Nas figuras 5.1 e 5.2 são apresentadas as análises probabilísticas da instância 50 e 100, respectivamente, do conjunto de testes A. A superioridade das versões G5 e G6 fica clara, sendo que no primeiro caso G5 possui uma vantagem sobre G6 e no segundo, ambas as versões competem pelo melhor desempenho. O pior desempenho é observada pelas versões G4 e G3, sendo que no primeiro caso a versão G4 conseguiu atingir o alvo tão poucas vezes que nem se consegue observar no gráfico e no segundo caso a versão G3 apresentou o pior desempenho.

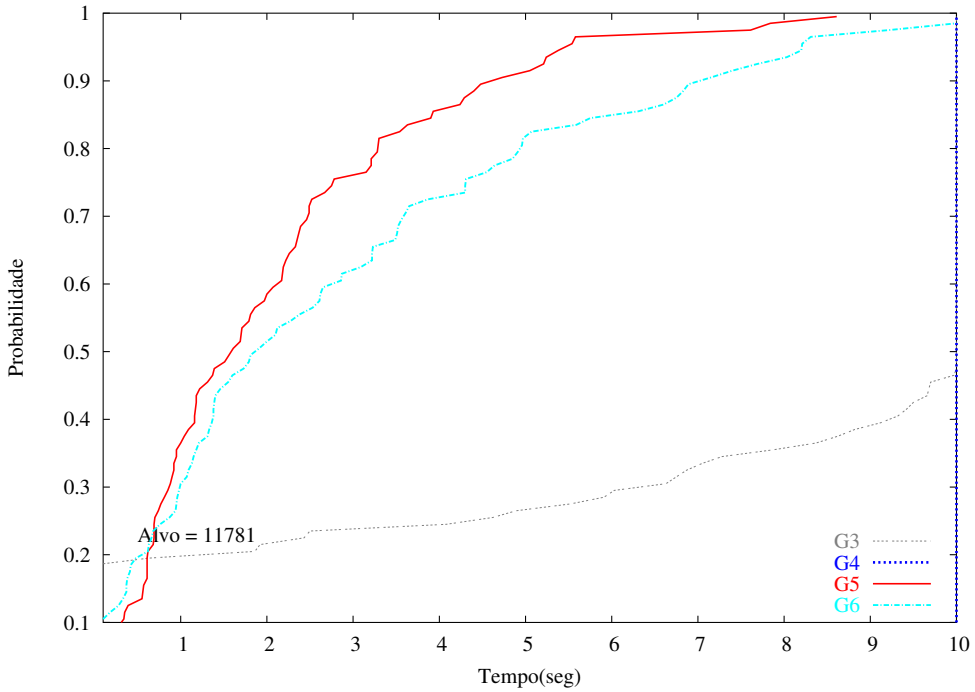

Figura 5.1: Análise probabilística das versões puras - instância 50 do conjunto de testes A

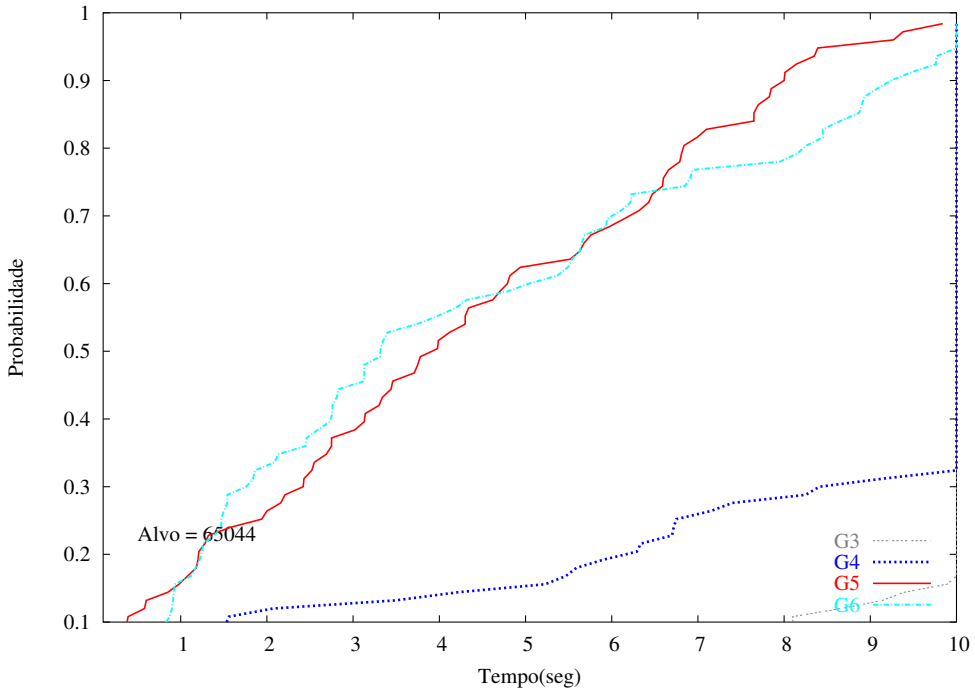

Figura 5.2: Análise probabilística das versões puras - instância 100 do conjunto de testes A

Notamos ainda que a versão G5, no primeiro caso, obteve no tempo aproximado de $t = 2$ segundos, uma probabilidade de convergência empírica de 50% e G6 bem próximo

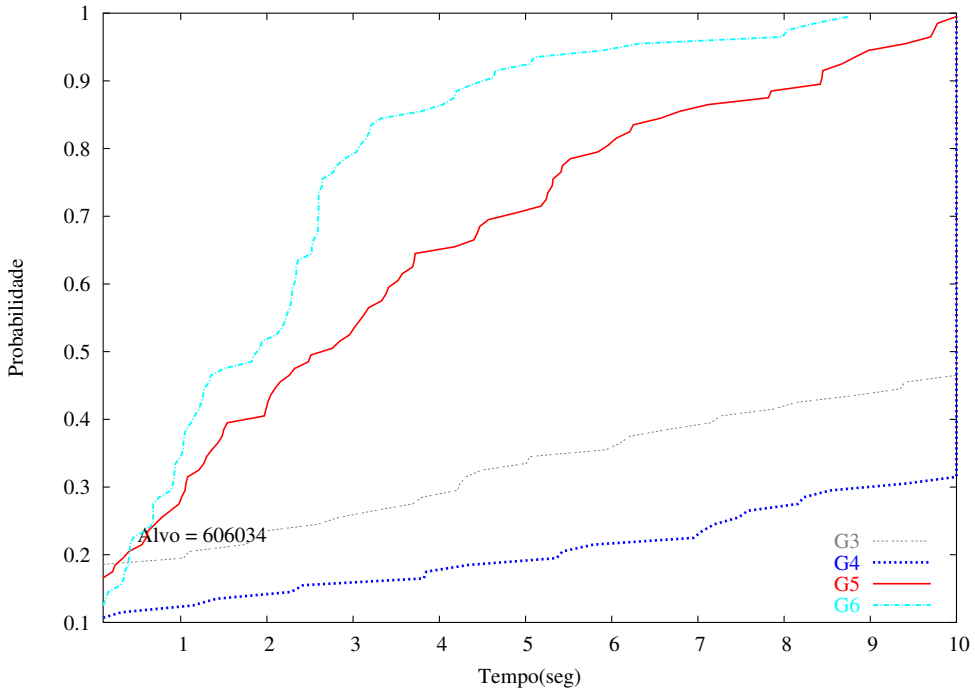

Figura 5.3: Análise probabilística das versões puras - instância 50 do conjunto de testes B

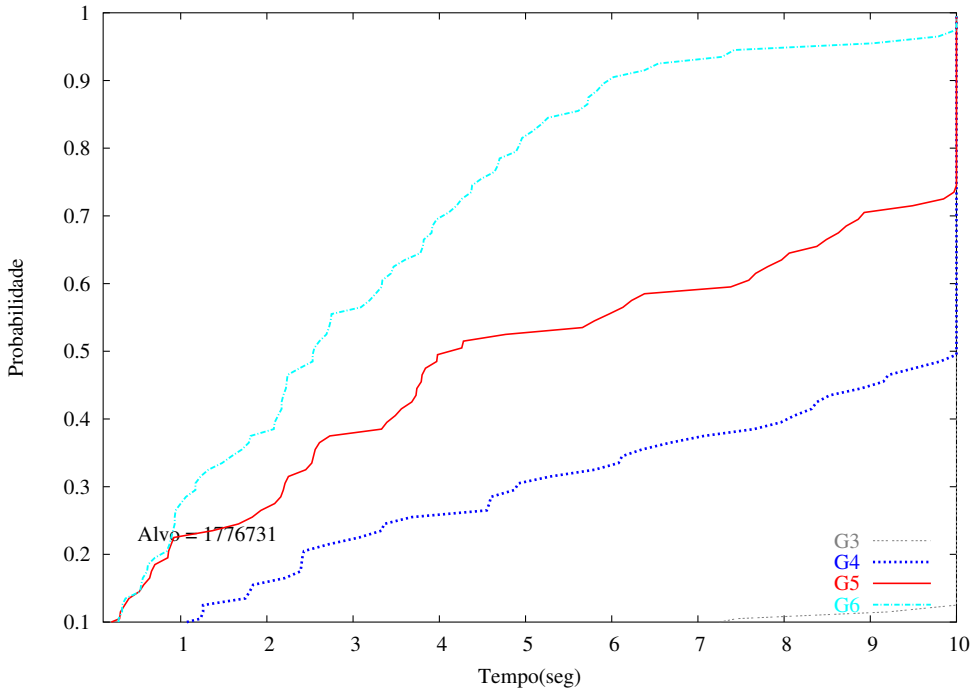

Figura 5.4: Análise probabilística das versões puras - instância 100 do conjunto de testes B

disso, enquanto as versões G3 apenas 20% e G4 menos de 1% (que nem se consegue observar). No segundo caso, no tempo aproximado de $t = 5$, G5 e G6 já apresentam 60%

de convergencia, enquanto G4 apenas cerca de 15% e G3 0%.

Nas figuras 5.3 e 5.4 são apresentadas as análises probabilísticas da instância 50 e 100, respectivamente, do conjunto de testes B. Podemos notar que em ambos os casos a versão G6 obteve uma clara superioridade, ficando a versão G5 com o segundo melhor desempenhos e G4 e G3 alternaram a pior colocação.

Em uma segunda etapa executamos a análise probabilística empírica com as versões G6 e G7 para comparar a eficiência do módulo de RC. O resultado dessas análises pode ser visto nas figuras de 5.5 à 5.8. Nestas simulações o valor alvo considerado é de melhor qualidade do que os das simulações anteriores, já que são submetidas duas versões que se mostraram eficientes nos testes preliminares.

O objetivo nesta análise é mostrar que as versões com iterações mais "pesadas", como é o caso do G7, muitas vezes atingem um valor alvo em tempos computacionais menores do que versões mais "leves", como é o caso do G6.

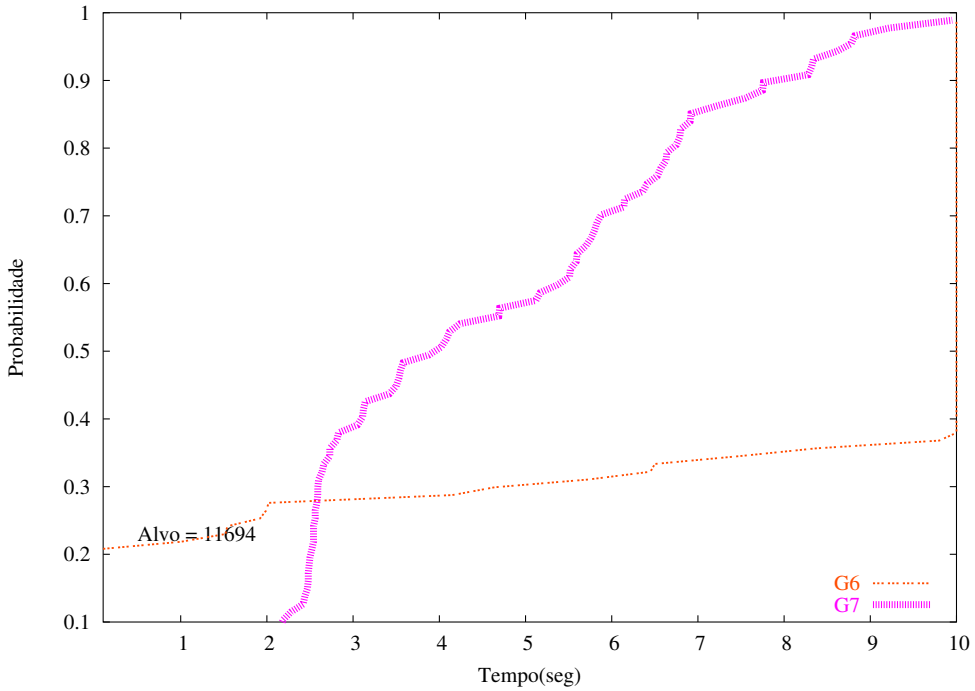

Figura 5.5: Análise probabilística das versões G6 e G7 da instância 50 do conjunto de testes A

Nas figuras de 5.5 à 5.8 podemos observar que a versão mais sofisticadas (G7) demanda, na maior parte dos casos, um tempo computacional menor ou similar à versão G6. Podemos observar ainda que a versão G7 sempre atinge o valor alvo, enquanto a

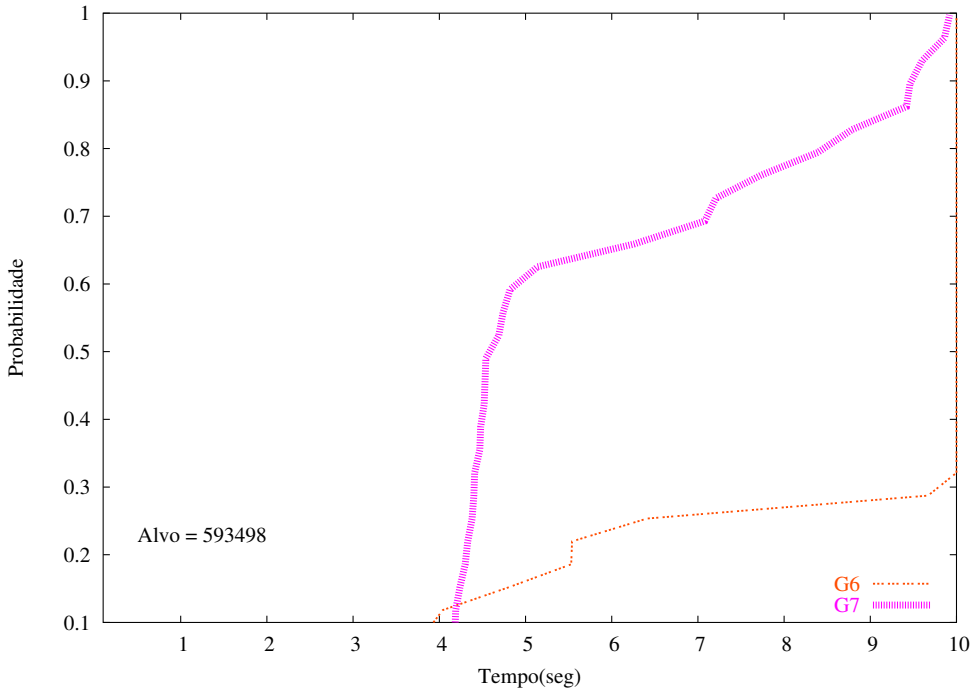

Figura 5.6: Análise probabilística das versões G6 e G7 da instância 50 do conjunto de testes B

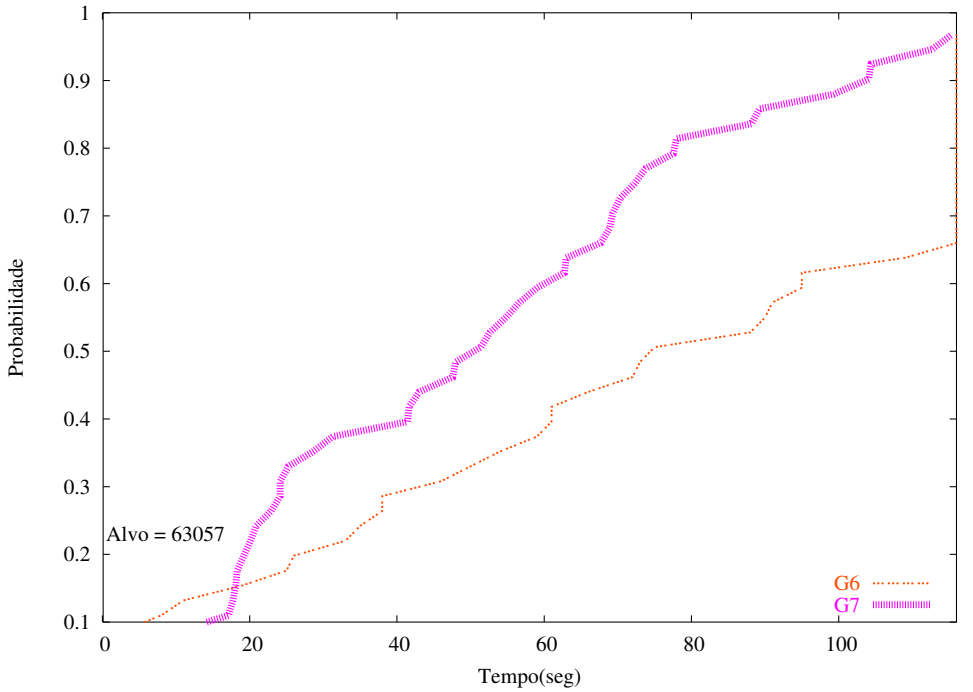

Figura 5.7: Análise probabilística das versões G6 e G7 da instância 100 do conjunto de testes A

versão G6 consegue atingi-lo apenas em 80% das execuções.

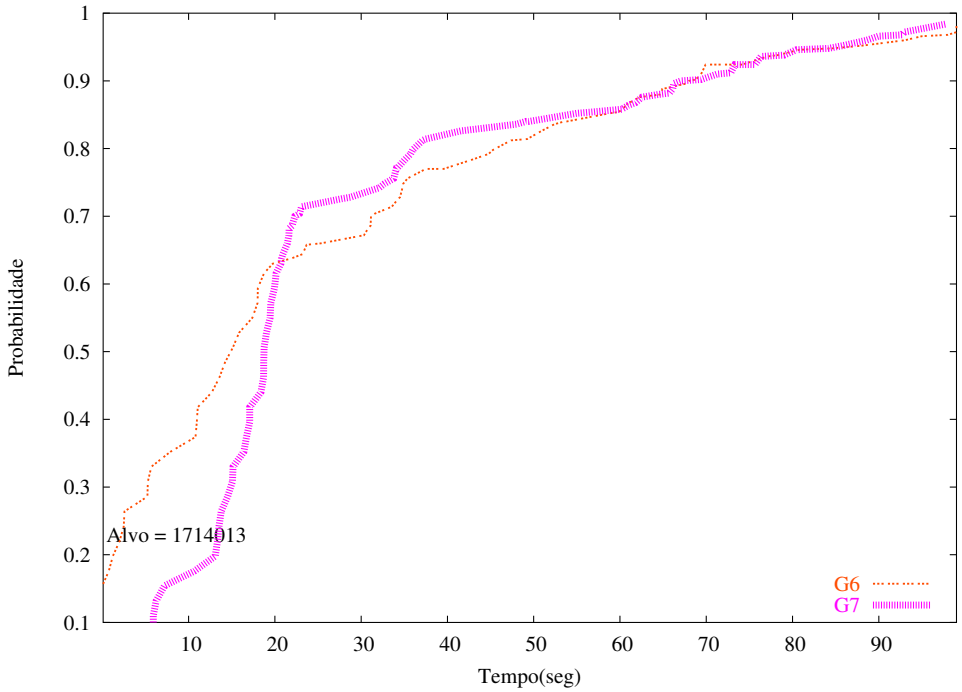

Figura 5.8: Análise probabilística das versões G6 e G7 da instância 100 do conjunto de testes B

O melhor desempenho da versão com reconexão por caminhos (G7), quando comparada à versão pura (G6), fica evidente nas figuras 5.5 à 5.8. Na figura 5.5, usando o alvo 11.694, observamos que o tempo exigido para o G7 obter uma convergência empírica de 100% nas 100 execuções efetuadas é de cerca de 10 segundos, enquanto neste mesmo tempo, a versão sem reconexão por caminhos (G6) atinge uma convergência menor que 40%.

Na figura 5.6, usando o alvo 593.498, observamos que o desempenho de G6 é ainda menos eficiente diante da versão G7, conseguindo em um tempo de cerca de 10 segundos uma convergência empírica de pouco mais de 30%, enquanto neste mesmo tempo, a versão G7 converge em 100% das execuções.

Na figura 5.7, usando o alvo 63.057, é o caso onde a versão G6 apresenta o seu segundo melhor desempenho dentre os casos apresentados, pois consegue convergência de cerca de 65% enquanto G7 continua conseguindo convergir em 100% das execuções.

Já na figura 5.8, usando o alvo 1.714.013, é o caso onde a versão G6 apresenta o seu melhor desempenho, sendo bastante competitiva com a verção G7 em termos de convergência empírica, conseguindo convergir em quase 100% das execuções, tal como

G7.

5.8 Evolução das soluções GRASP

Para termos uma melhor visão das soluções geradas pela heurística GRASP a cada iteração, apresentamos nesta seção alguns gráficos que ilustram a curva com valores associados à solução de cada iteração de uma heurística GRASP em relação ao *best* de cada instância (*best* é a melhor solução da instância considerando todas as heurísticas aqui analisadas).

No eixo da coordenada X encontra-se o número da iteração (que varia de 1 à 200) e no eixo da coordenada Y encontra-se o valor da solução alcançado por cada versão apresentada.

Os gráficos gerados são referentes às versões puras (G1, G2, G3, G4, G5 e G6) e às versões adaptativas GAdapt4, GAdapt10 e GAdapt12.

A ilustração dos gráficos das demais versões que incorporam módulo de *RC* não faz sentido, pois por apresentarem o módulo de RC apenas ao final da execução da heurística GRASP, estas apresentariam um gráfico idêntico ao da versão G6, tendo apenas uma evolução maior no final do gráfico, quando o módulo *RC* é ativado.

Quanto às versões adaptativas, apenas as versões que se destacaram na primeira fase de testes foram consideradas.

As figuras de 5.9 à 5.29 apresentam a evolução das soluções por iteração das versões puras (de G1 à G6) para o conjunto de testes A.

Nas figuras 5.9 e 5.10 são apresentadas as evoluções das soluções por iteração das versões G1 e G2, ambas usam a BL1, onde podemos constatar que sempre ficam distantes da solução *best* da instância.

Nas figuras 5.11 e 5.12 são apresentadas as evoluções das soluções por iteração das versões G3 e G4, ambas usam a BL2, onde verificamos que as soluções geradas por estas ficam muito mais próximas da solução *best* do que as soluções geradas pelas versões G1 e G2.

Nas figuras 5.13 e 5.14 são apresentadas as evoluções das soluções por iteração das

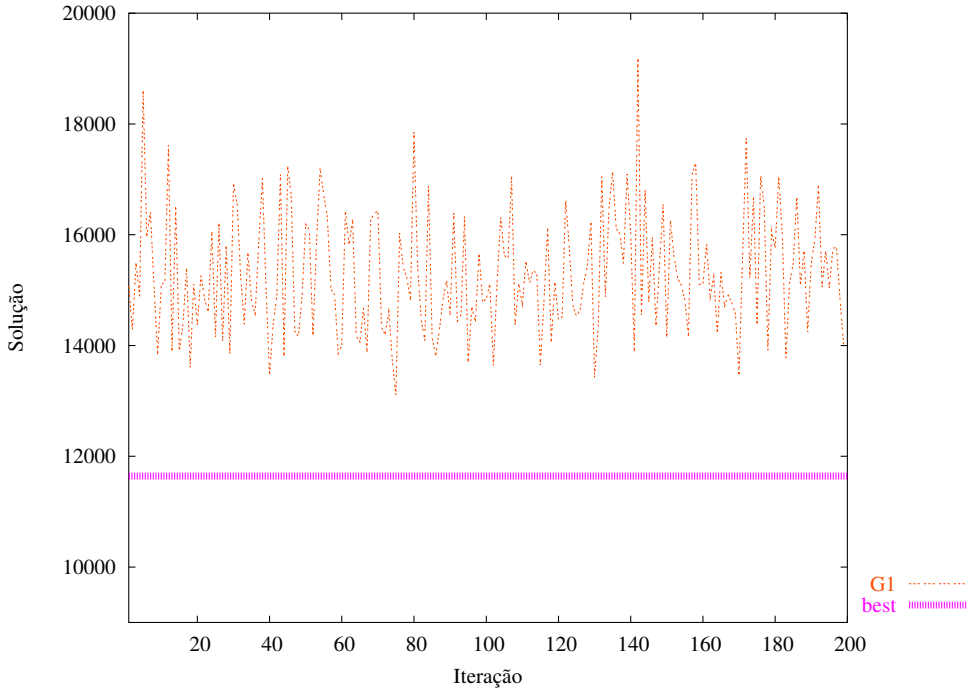

Figura 5.9: Solução por iteração - versão G1 - instância 50 do conjunto de testes A

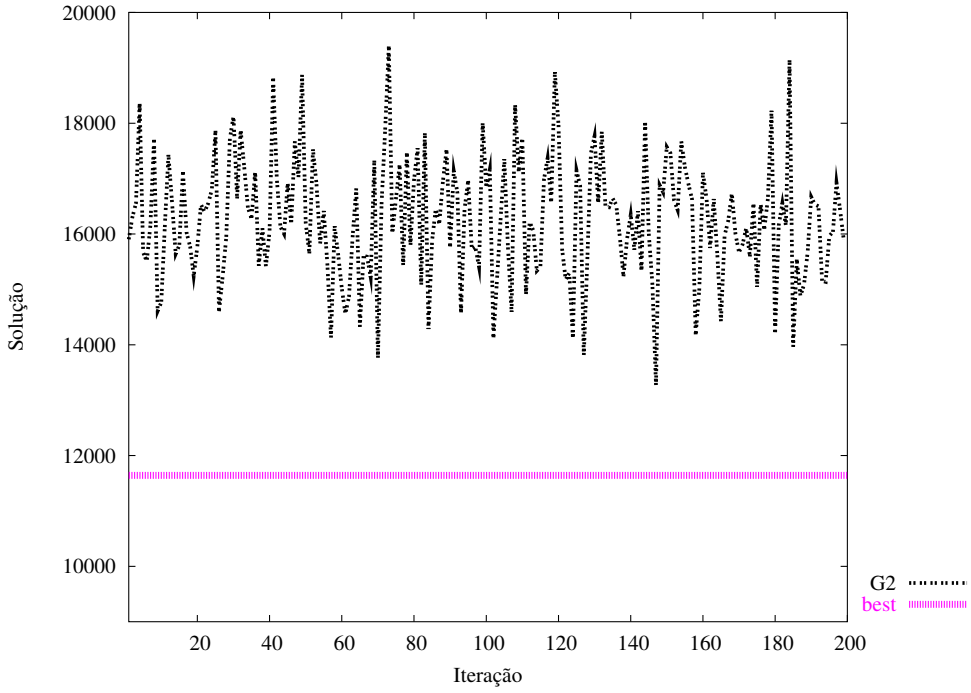

Figura 5.10: Solução por iteração - versão G2 - instância 50 do conjunto de testes A

versões G5 e G6, ambas usam a BL3, onde verificamos uma maior proximidade dessas versões em relação ao *best* do que para as demais versões GRASP puro (G1, G2, G3 e G4). Essas figuras mostram ainda que as versões G5 e G6, ao contrário de G1, G2, G3 e

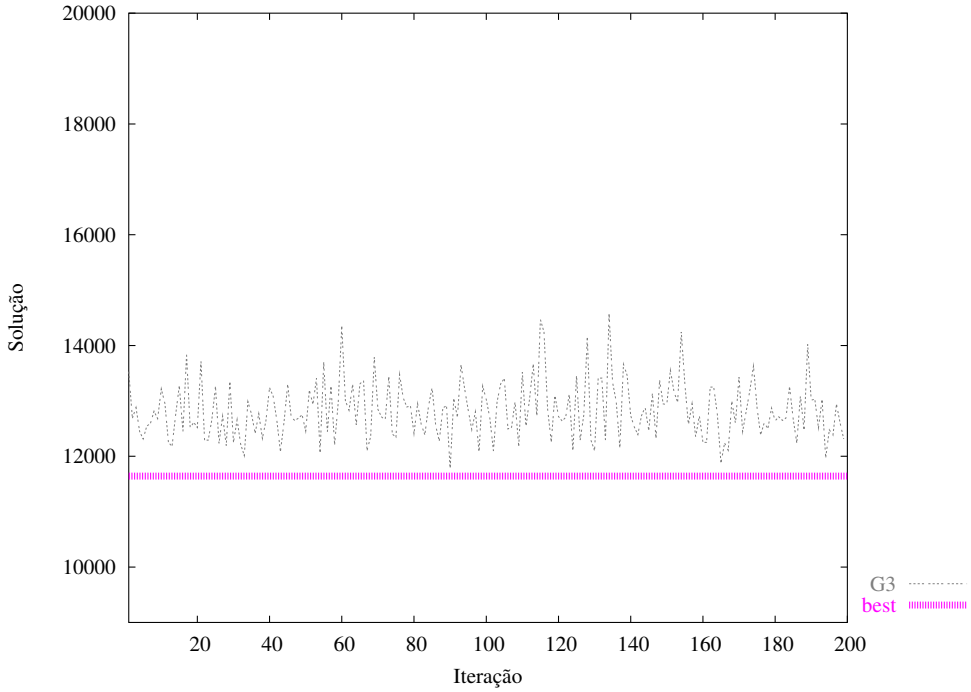

Figura 5.11: Solução por iteração - versão G3 - instância 50 do conjunto de testes A

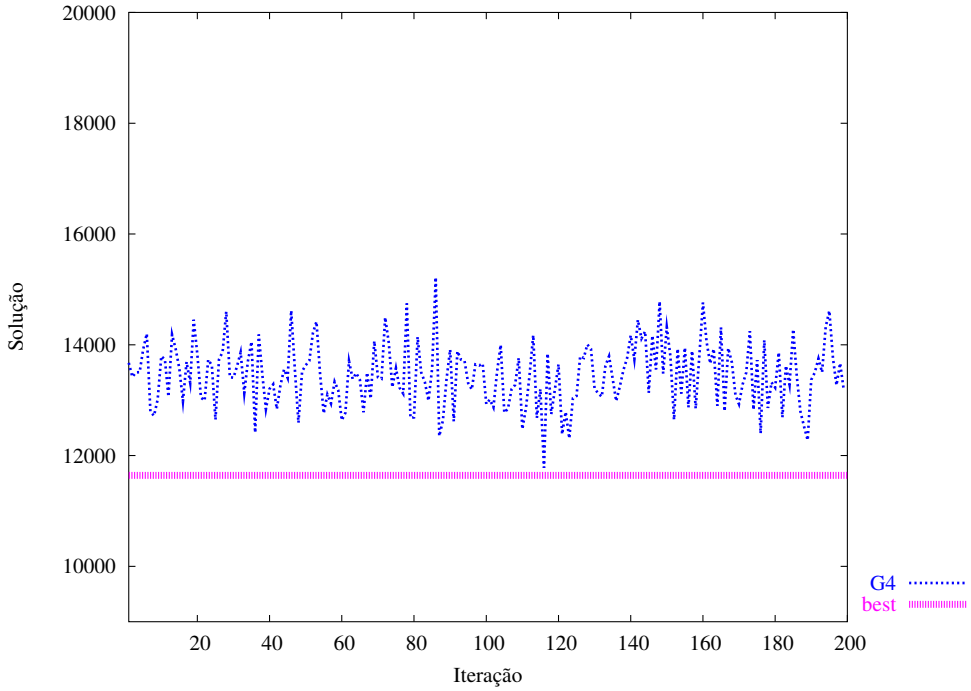

Figura 5.12: Solução por iteração - versão G4 - instância 50 do conjunto de testes A

G4, atingem o valor *best* em algumas iterações.

Esta comparação é melhor verificada na figura 5.15, onde todos os resultados são

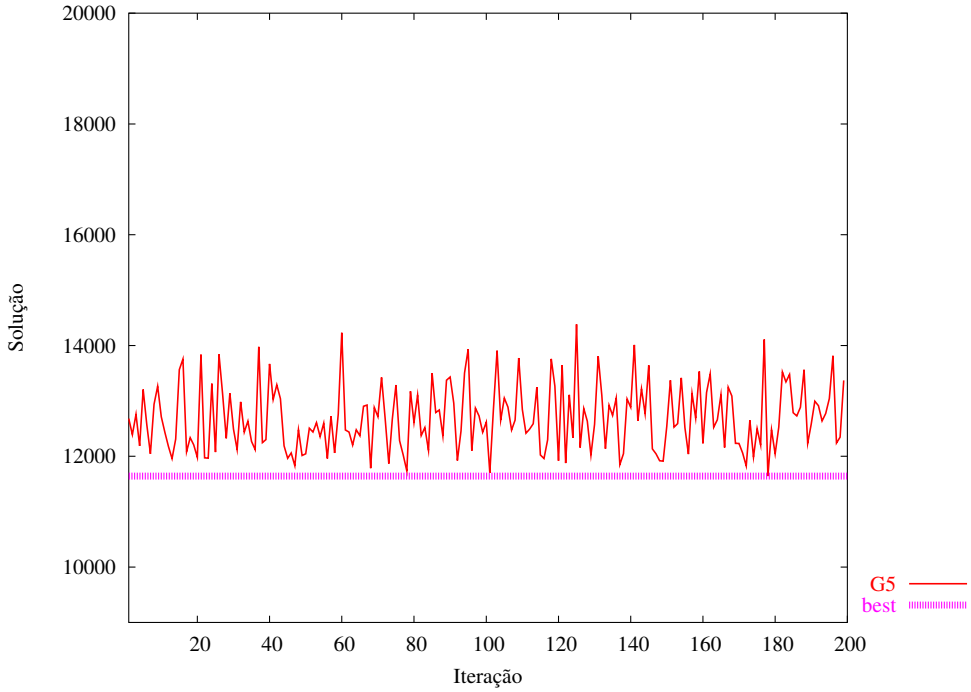

Figura 5.13: Solução por iteração - versão G5 - instância 50 do conjunto de testes A

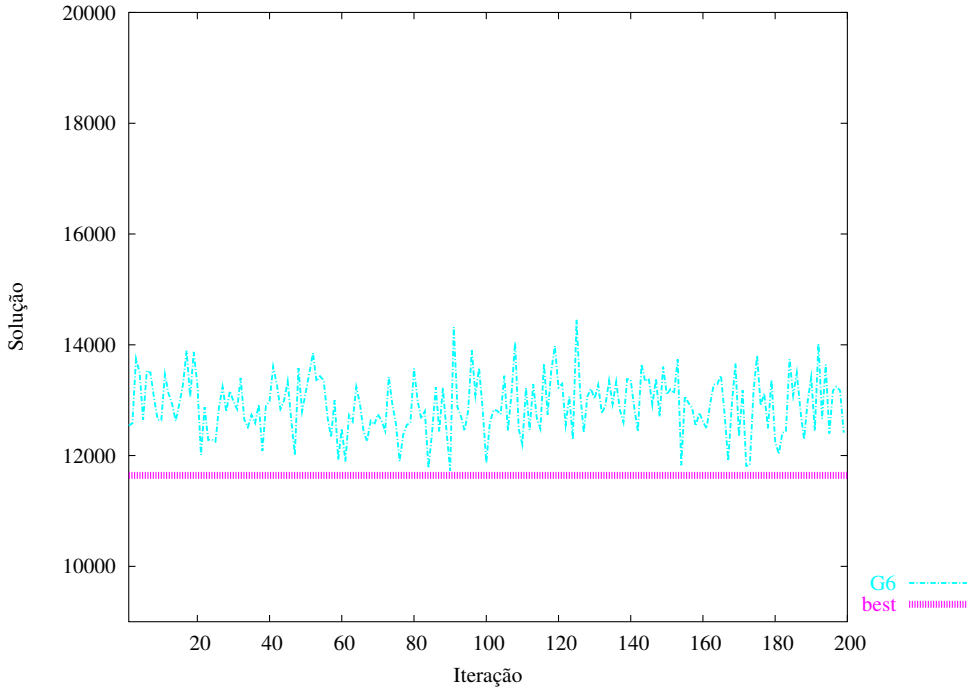

Figura 5.14: Solução por iteração - versão G6 - instância 50 do conjunto de testes A

apresentados juntos.

A mesma análise é feita à seguir para a instância de 100 poços (vértices) do conjunto

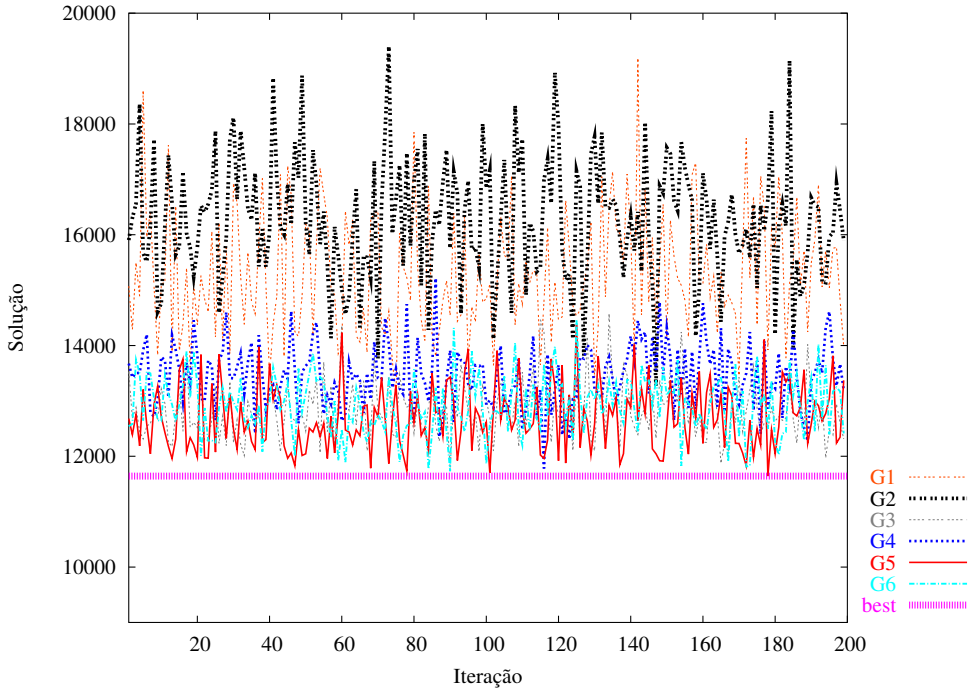

Figura 5.15: Solução por iteração das versões puras - instância 50 do conjunto de testes A

de testes A nas figuras de 5.16 à 5.22.

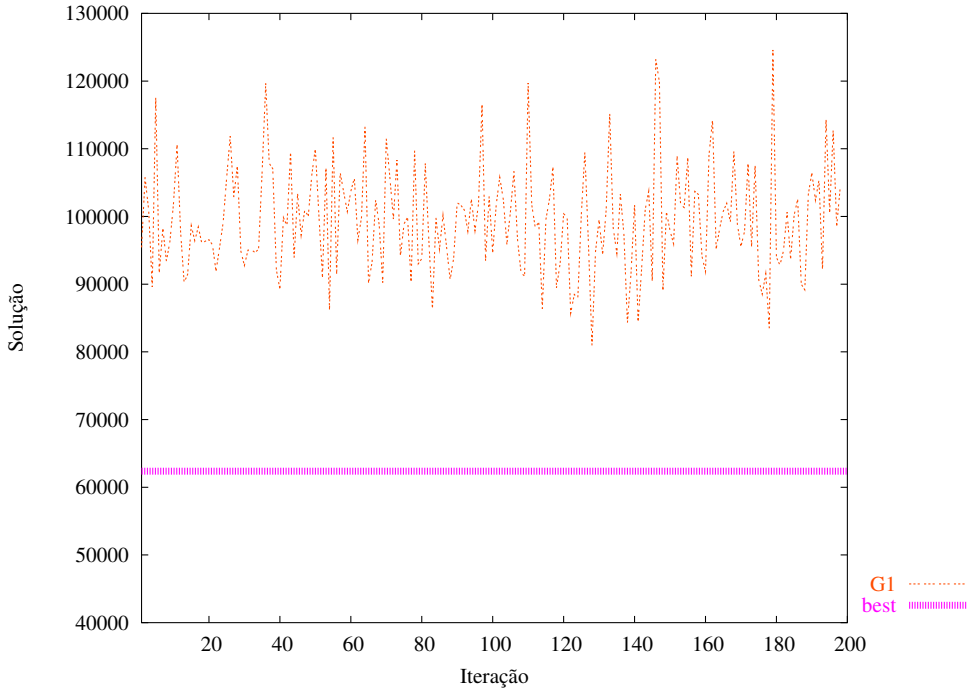

Figura 5.16: Solução por iteração - versão G1 - instância 100 do conjunto de testes A

Os resultados apresentados foram similares aos da instância de 50 poços, não havendo

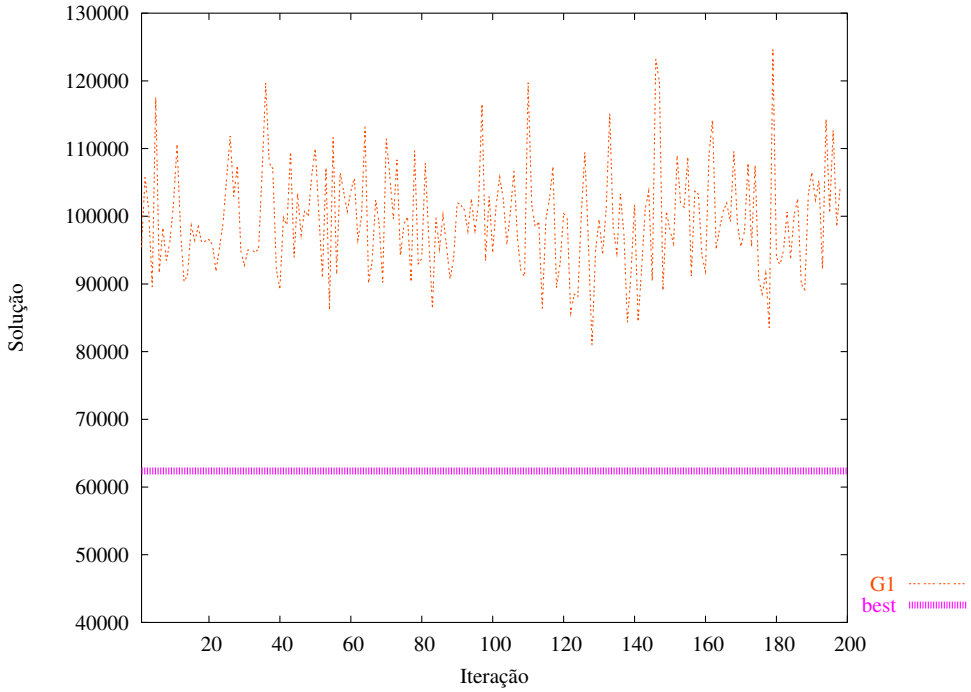

Figura 5.17: Solução por iteração - versão G2 - instância 100 do conjunto de testes A

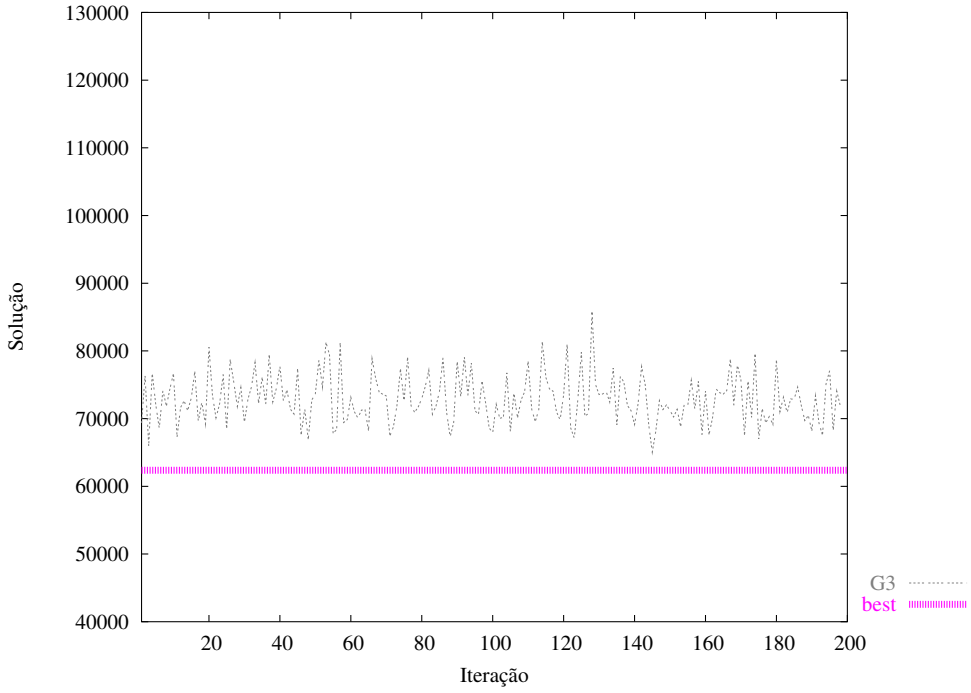

Figura 5.18: Solução por iteração - versão G3 - instância 100 do conjunto de testes A

anomalias nem exceções.

Nas figuras 5.23 à 5.29 são apresentados os resultados para a instância 200 do conjunto

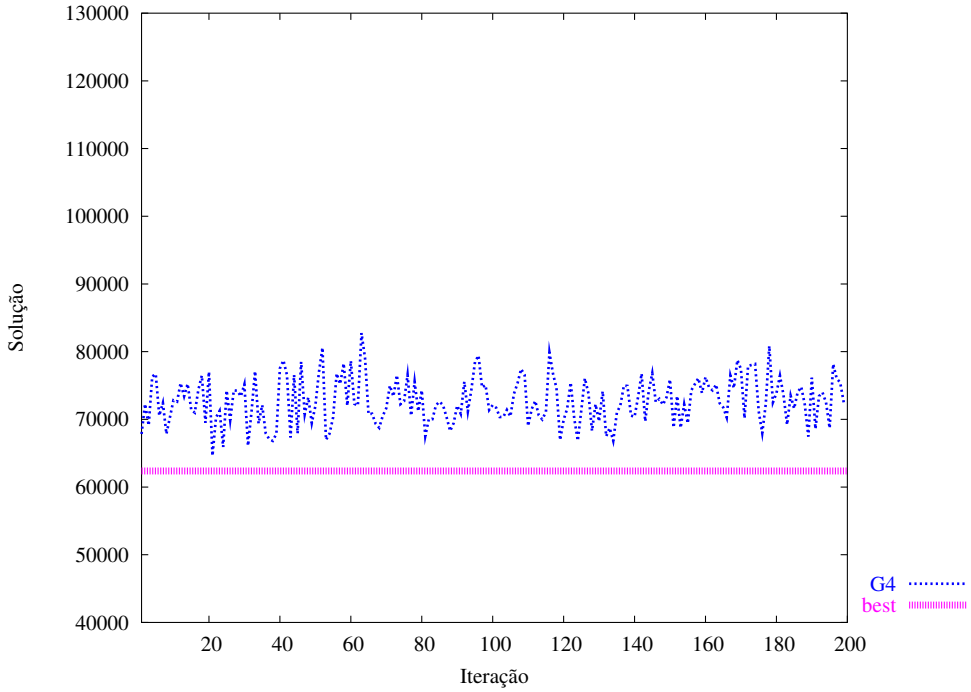

Figura 5.19: Solução por iteração - versão G4 - instância 100 do conjunto de testes A

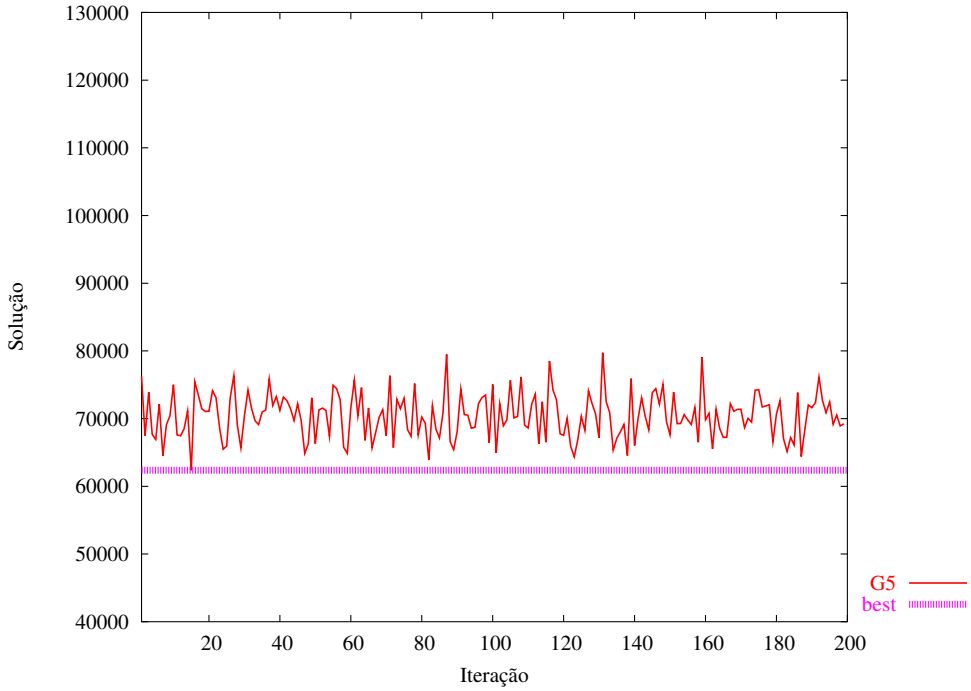

Figura 5.20: Solução por iteração - versão G5 - instância 100 do conjunto de testes A

de testes A.

Após verificar os resultados para as versões puras (G1, G2, G3, G4, G5 e G6) para

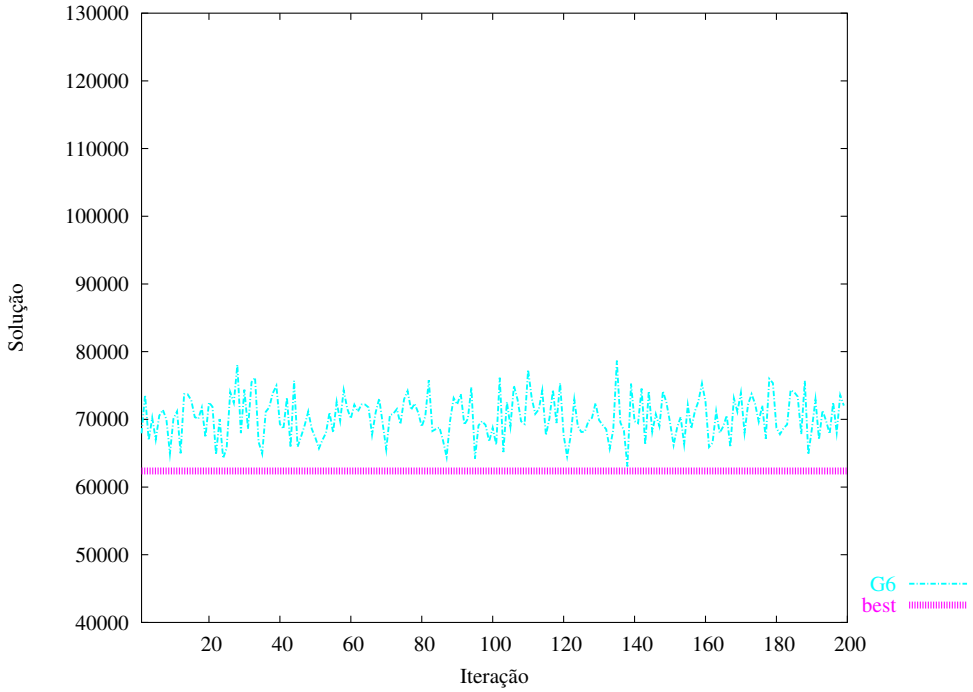

Figura 5.21: Solução por iteração - versão G6 - instância 100 do conjunto de testes A

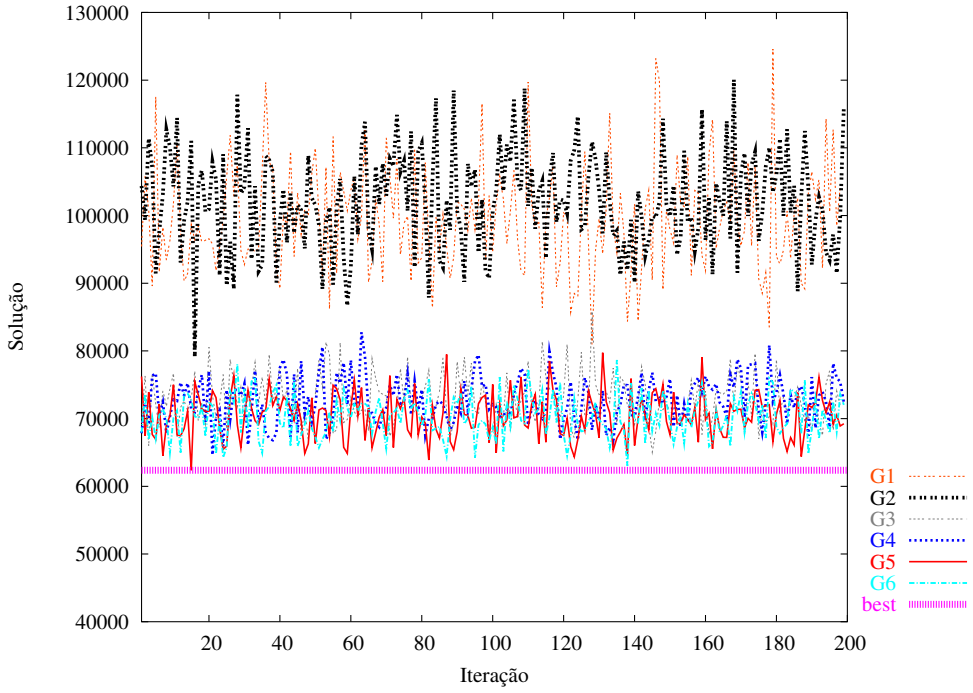

Figura 5.22: Solução por iteração das versões puras - instância 100 do conjunto de testes A

as instâncias de 50, 100 e 200 poços do conjunto de testes A podemos constatar que as versões se comportaram de forma similar para todos os casos, mostrando a superioridade

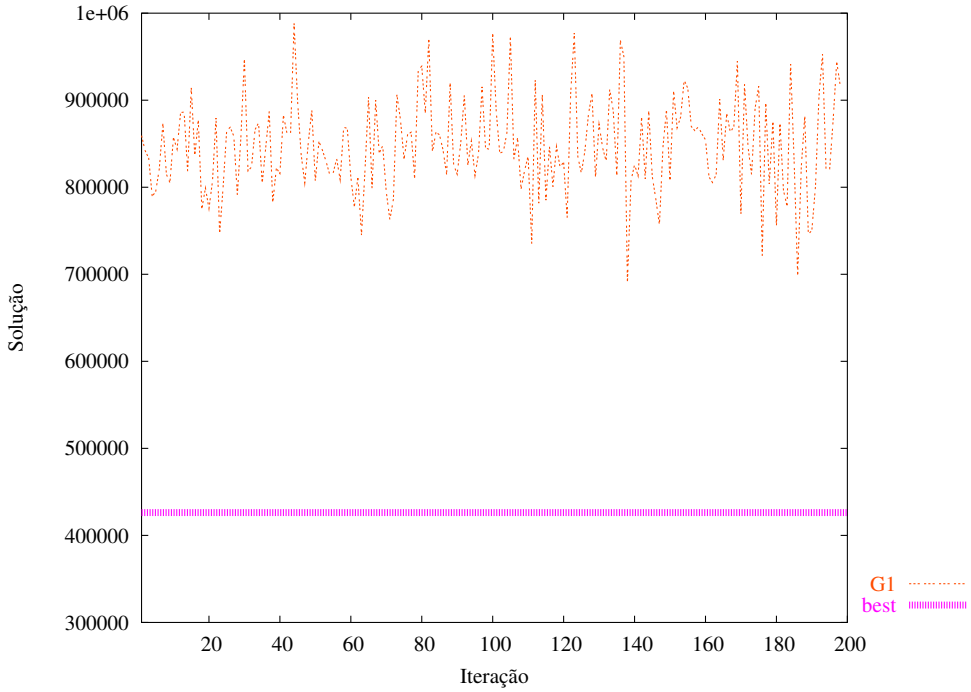

Figura 5.23: Solução por iteração - versão G1 - instância 200 do conjunto de testes A

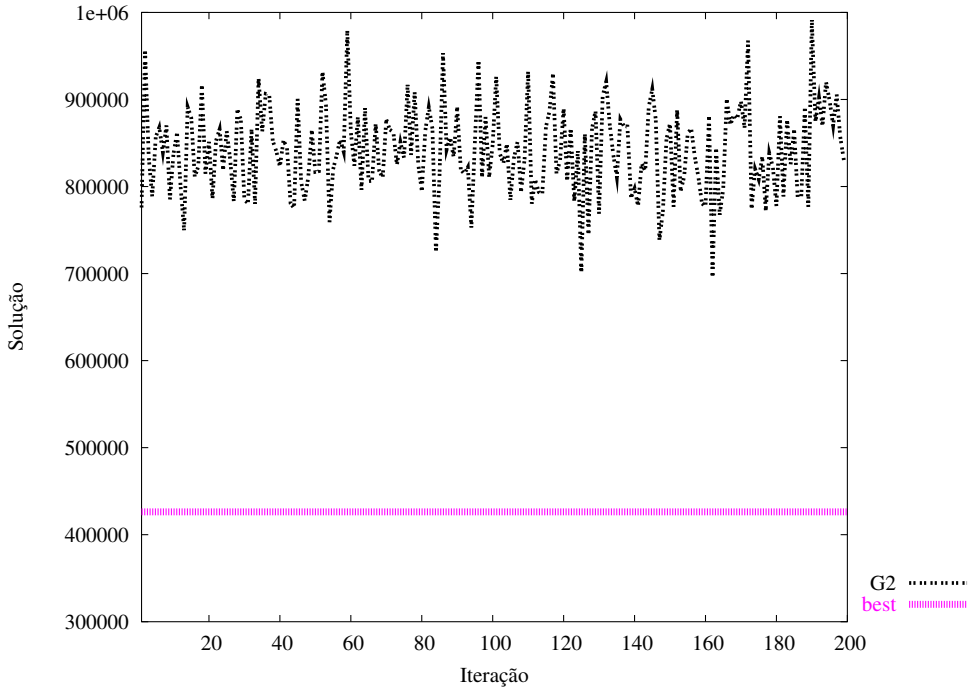

Figura 5.24: Solução por iteração - versão G2 - instância 200 do conjunto de testes A

da BL3 frente à BL1.

Os testes foram repetidos, agora para as instâncias do conjunto de testes B, e estão

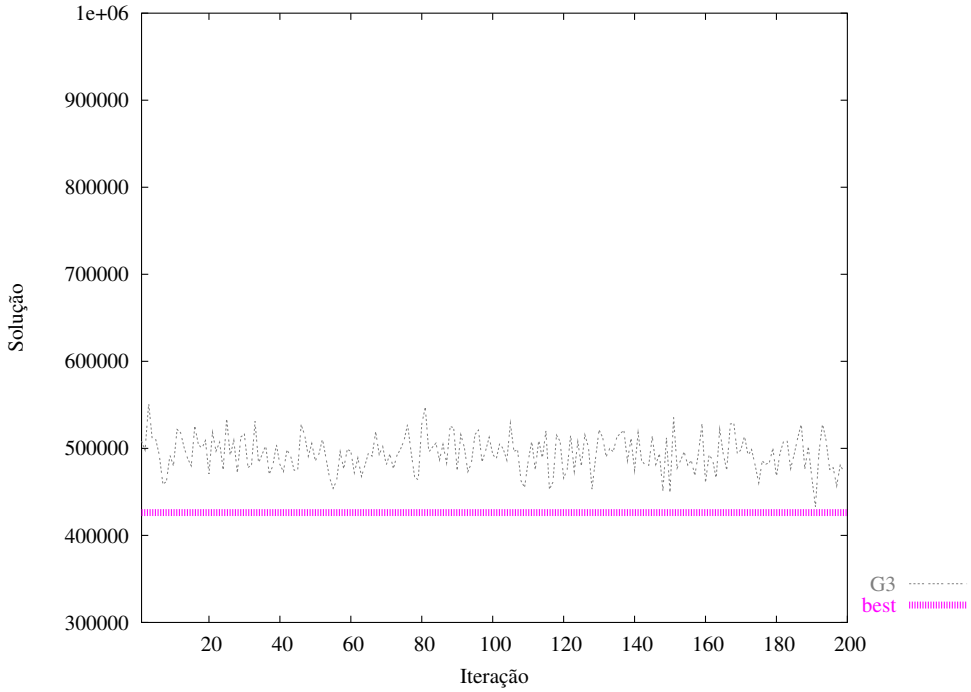

Figura 5.25: Solução por iteração - versão G3 - instância 200 do conjunto de testes A

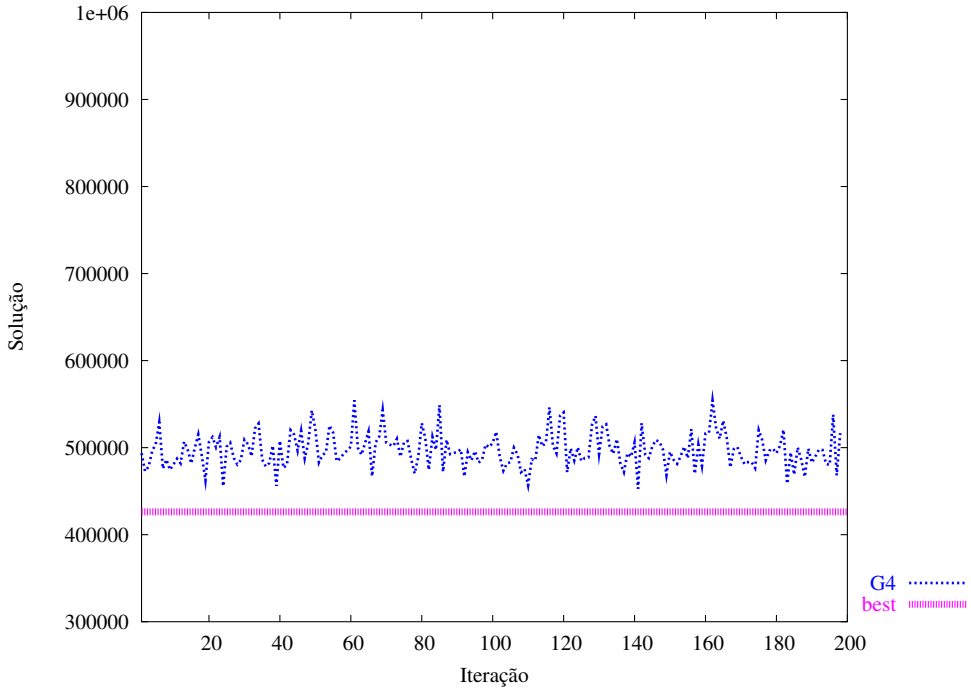

Figura 5.26: Solução por iteração - versões G4 - instância 200 do conjunto de testes A

apresentados nas figuras de 5.30 à 5.50.

Os testes para a instância de 50 poços (vértices), para o conjunto de testes B, estão

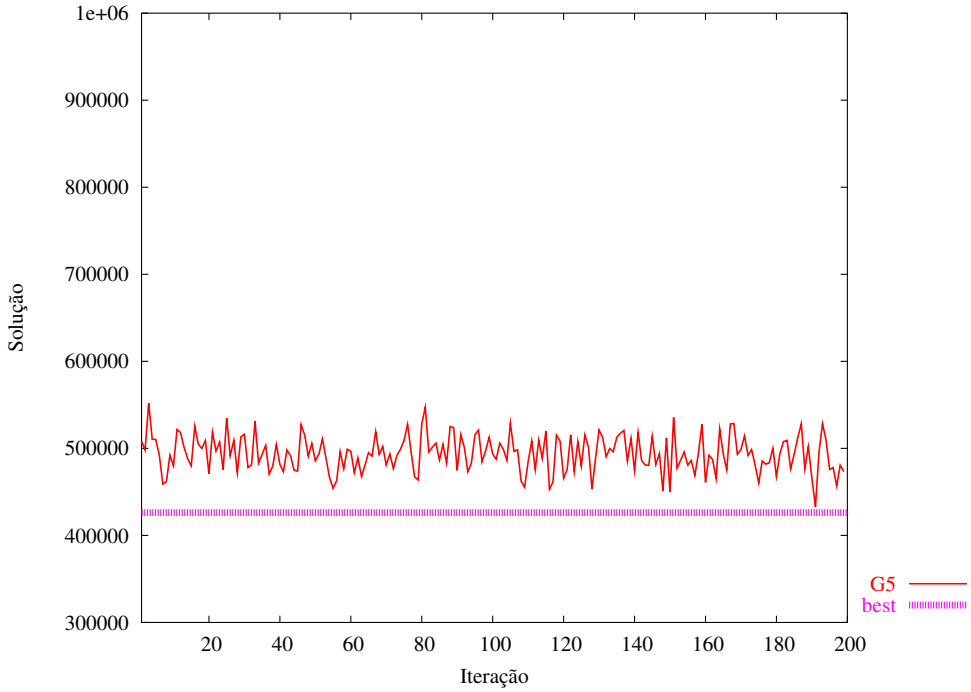

Figura 5.27: Solução por iteração - versão G5 - instância 200 do conjunto de testes A

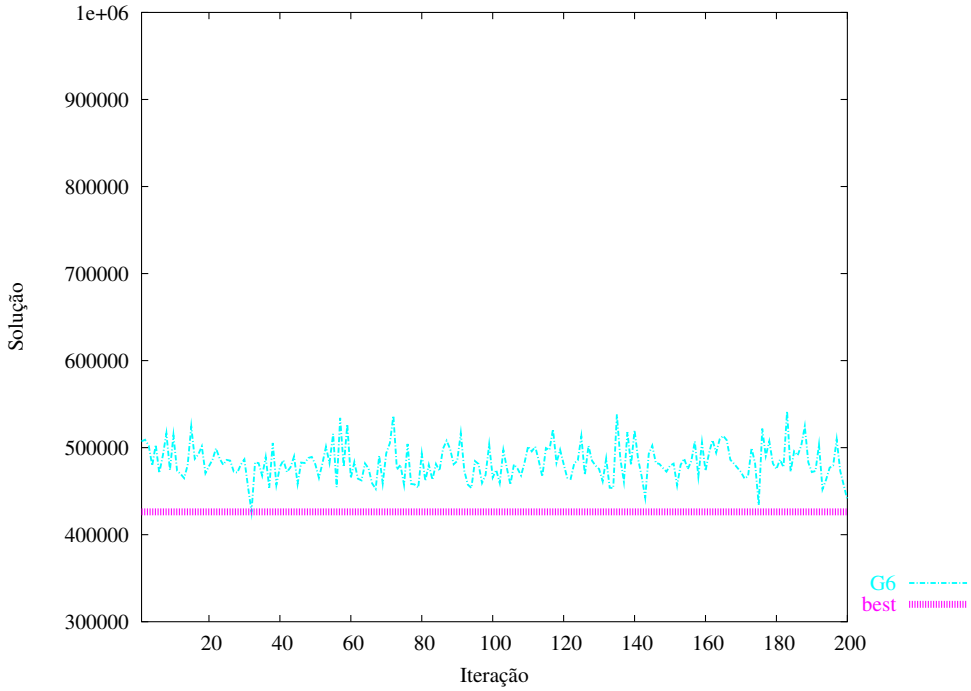

Figura 5.28: Solução por iteração - versão G6 - instância 200 do conjunto de testes A

apresentados nas figuras 5.30 à 5.36.

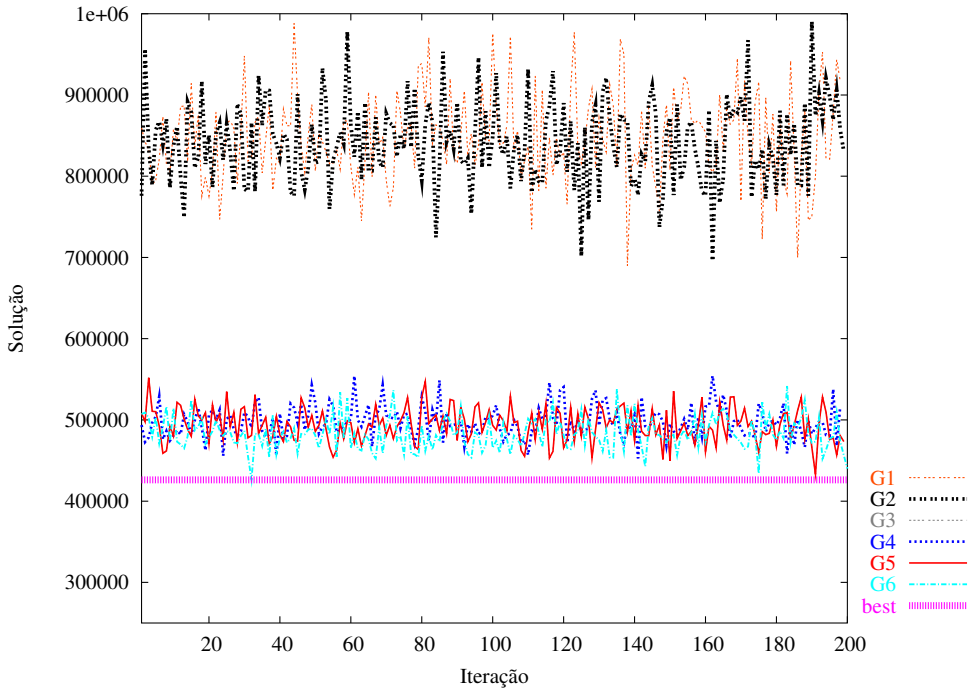

Figura 5.29: Solução por iteração das versões puras - instância 200 do conjunto de testes A

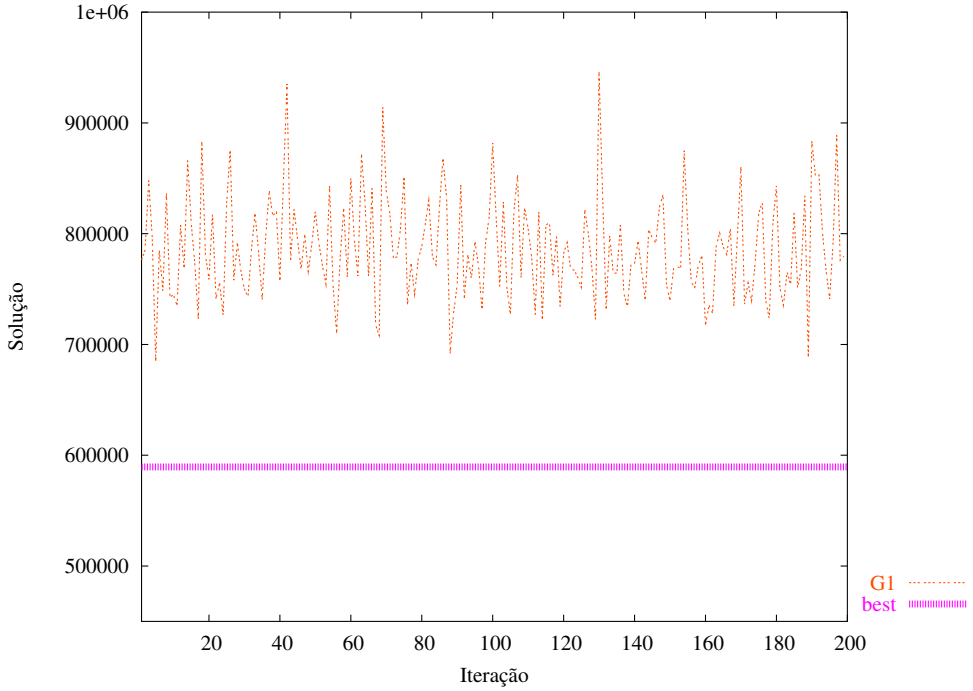

Figura 5.30: Solução por iteração - versão G1 - instância 50 do conjunto de testes B

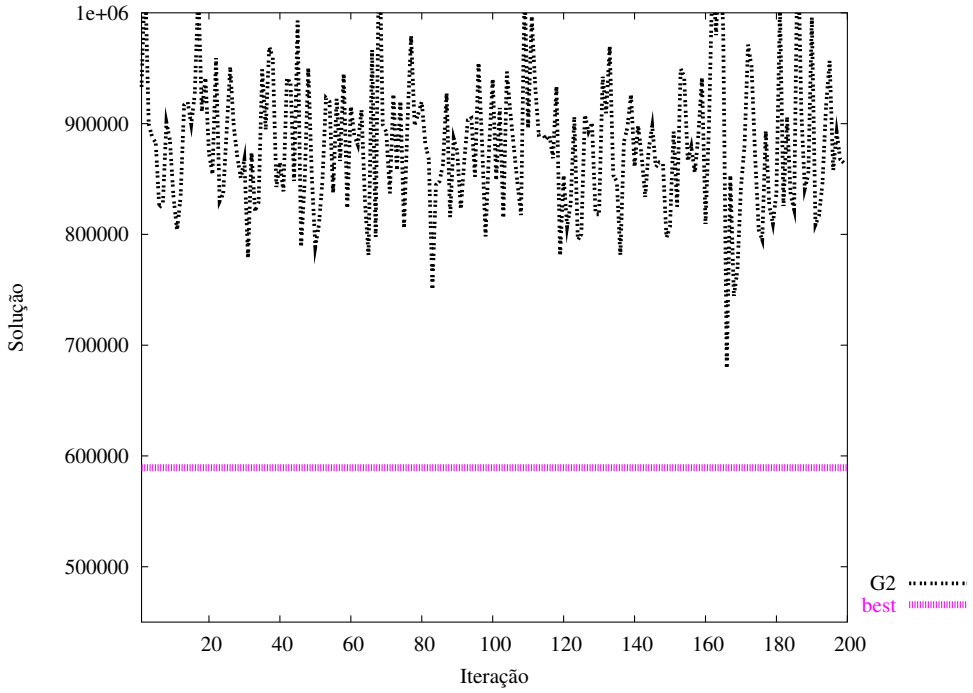

Figura 5.31: Solução por iteração - versão G2 - instância 50 do conjunto de testes B

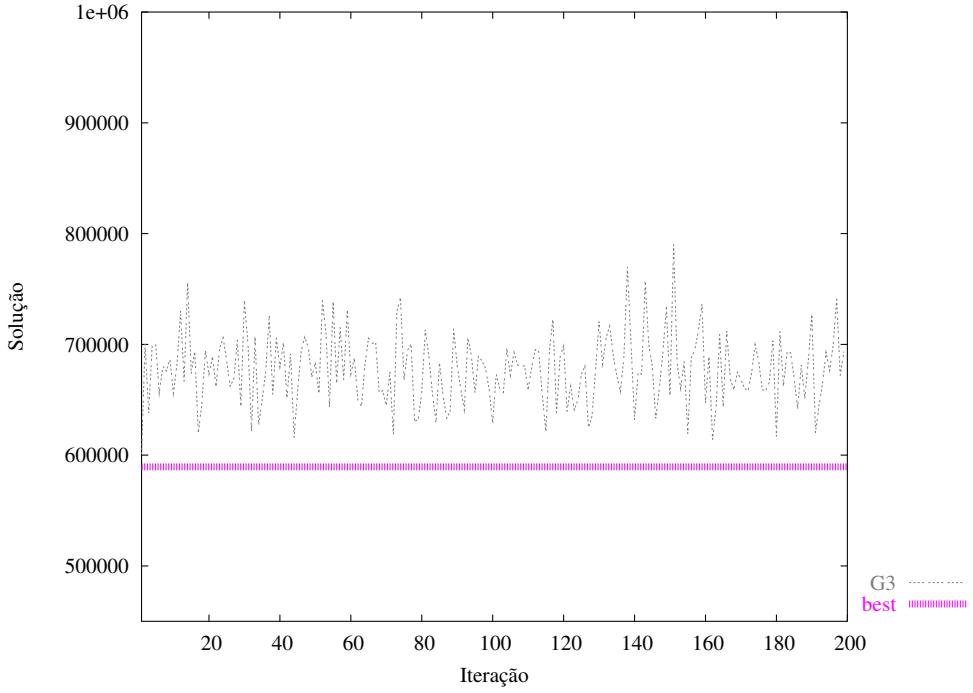

Figura 5.32: Solução por iteração - versão G3 - instância 50 do conjunto de testes B

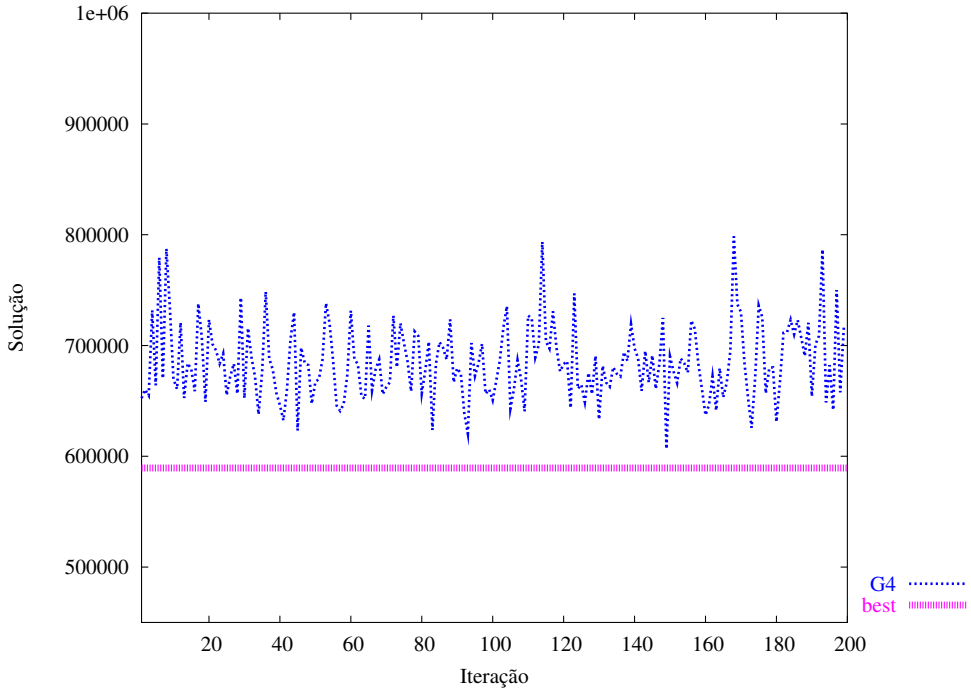

Figura 5.33: Solução por iteração - versão G4 - instância 50 do conjunto de testes B

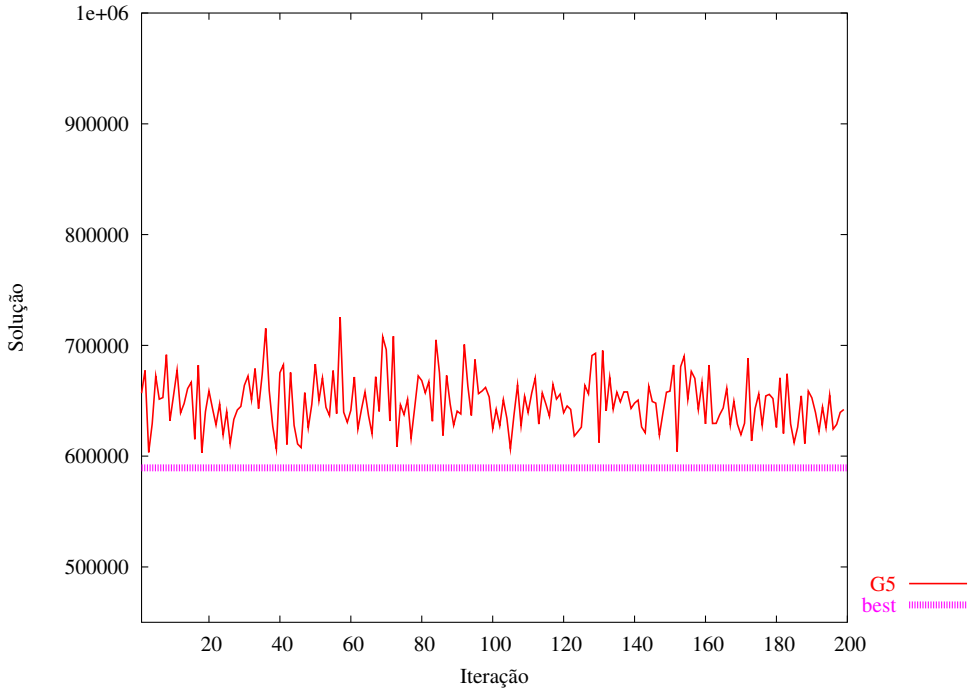

Figura 5.34: Solução por iteração - versão G5 - instância 50 do conjunto de testes B

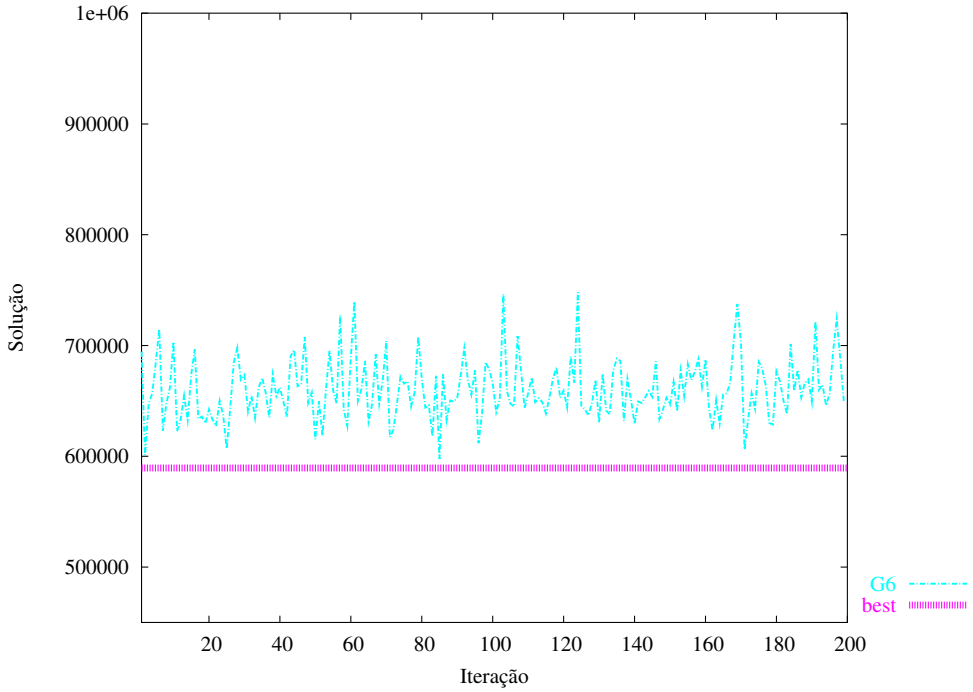

Figura 5.35: Solução por iteração - versão G6 - instância 50 do conjunto de testes B

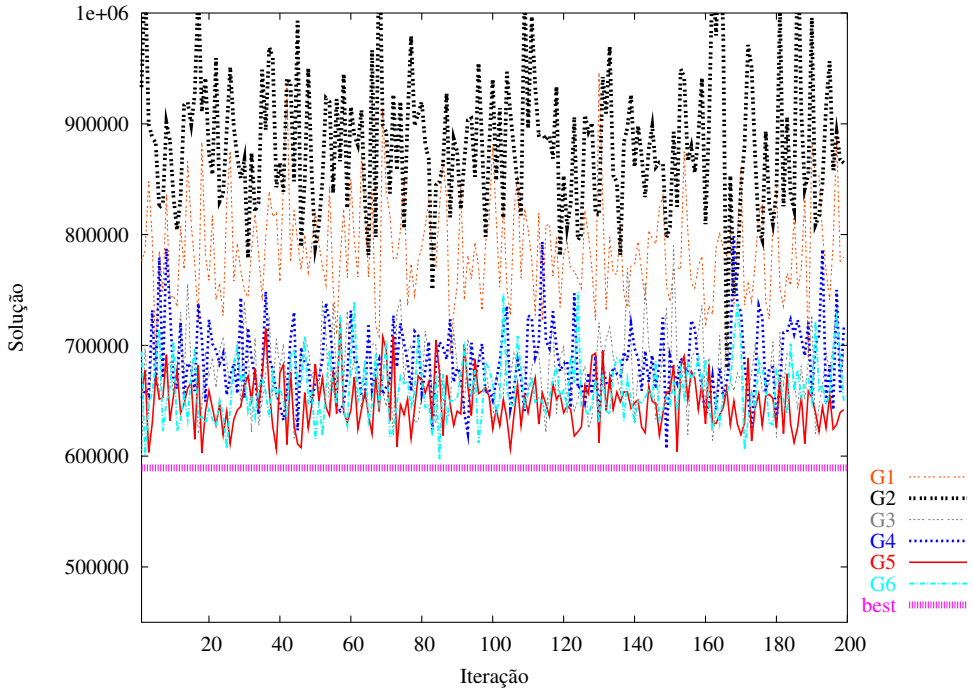

Figura 5.36: Solução por iteração das versões puras - instância 50 do conjunto de testes B

Os testes para a instância de 100 poços (vértices), para o conjunto de testes B, estão apresentados nas figuras 5.37 à 5.43.

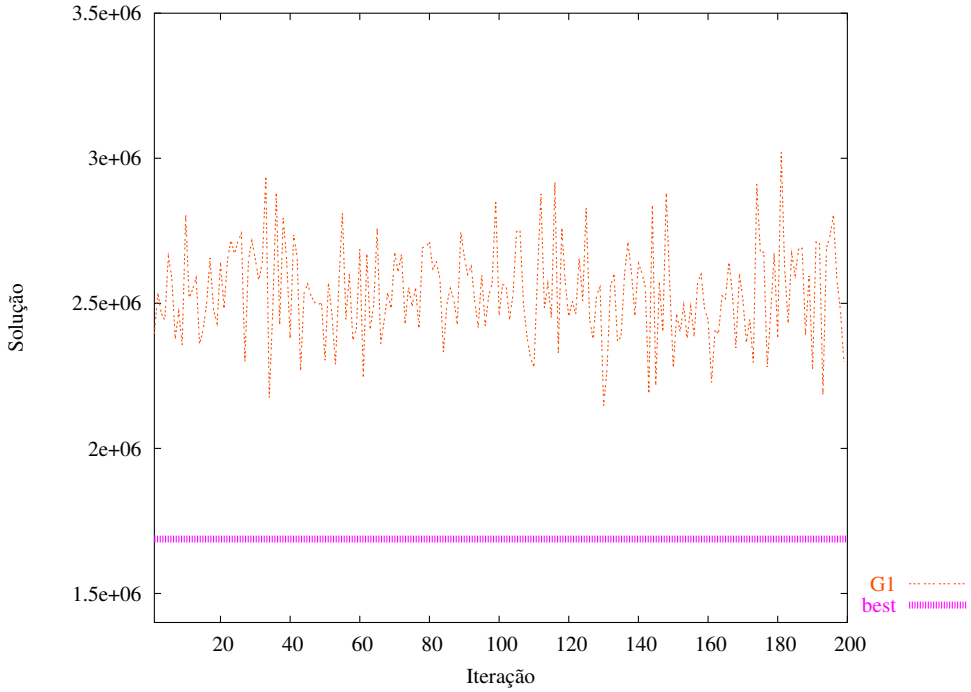

Figura 5.37: Solução por iteração - versão G1 - instância 100 do conjunto de testes B

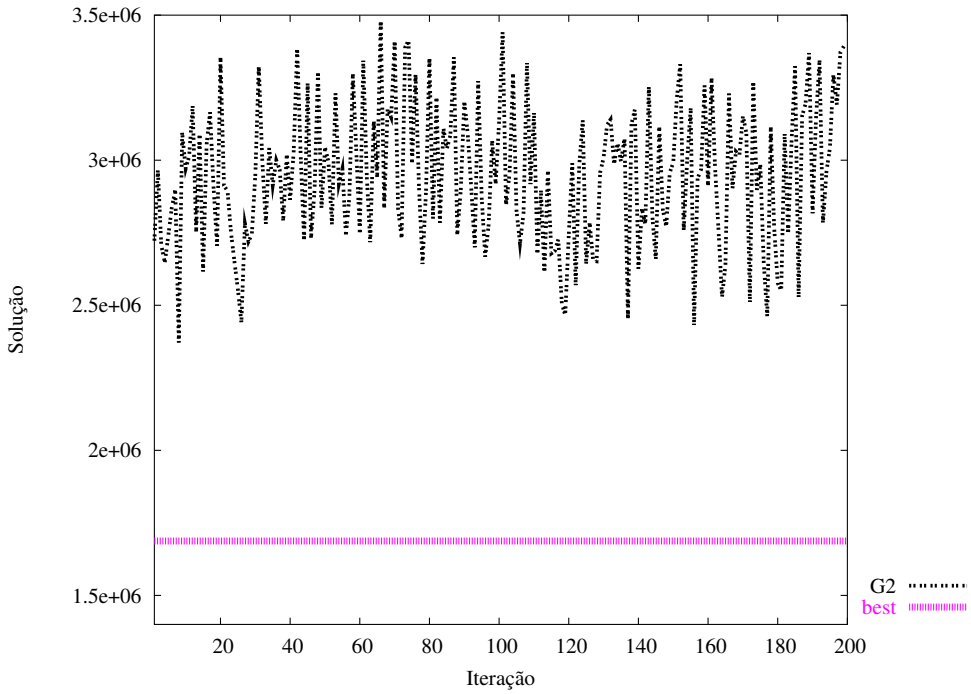

Figura 5.38: Solução por iteração - versão G2 - instância 100 do conjunto de testes B

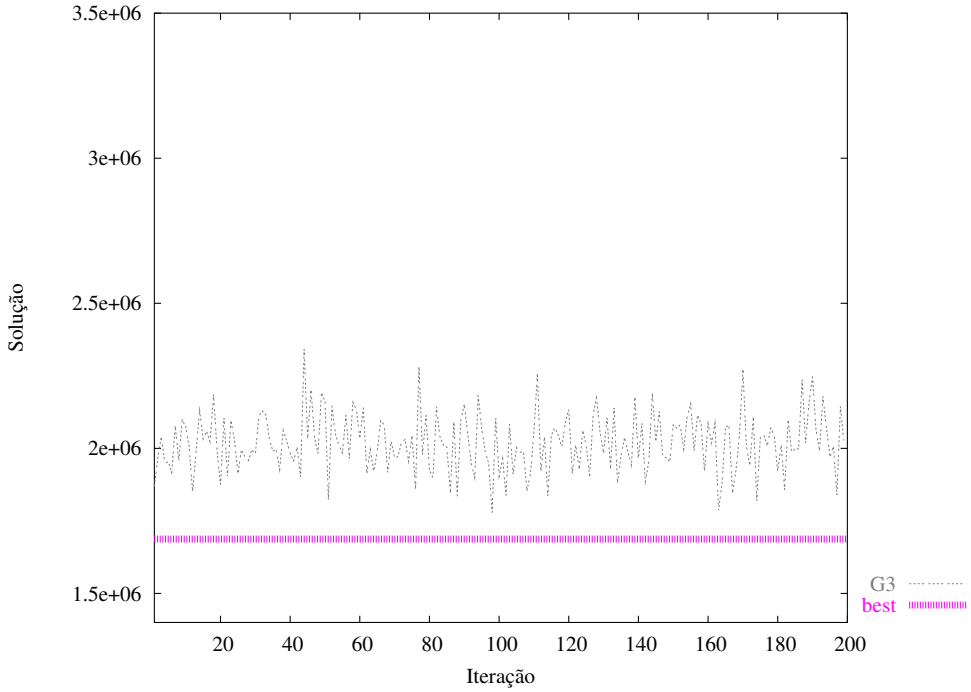

Figura 5.39: Solução por iteração - versão G3 - instância 100 do conjunto de testes B

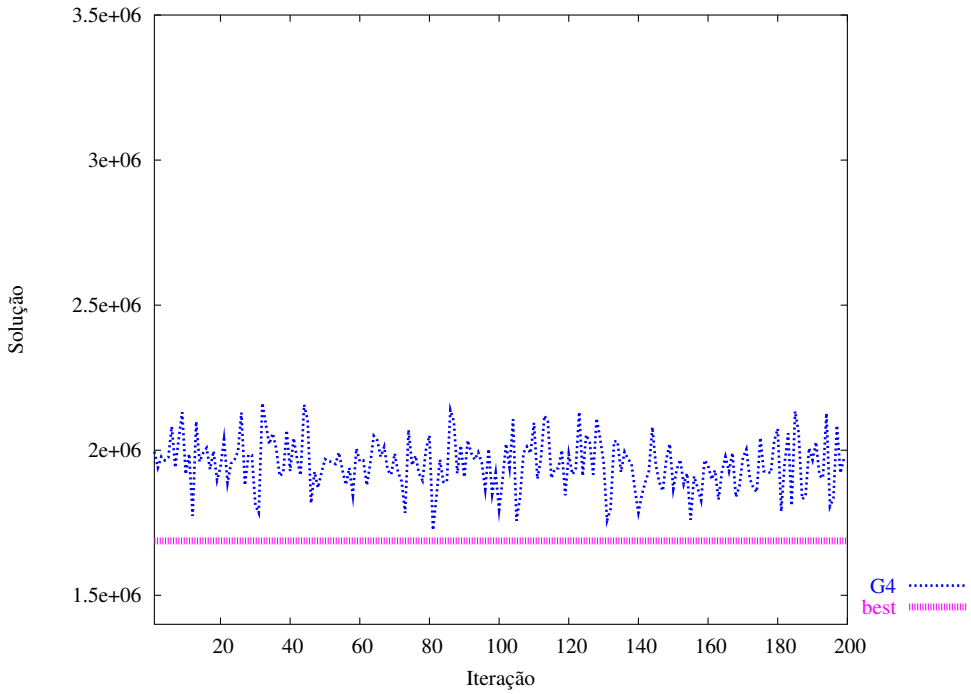

Figura 5.40: Solução por iteração - versão G4 - instância 100 do conjunto de testes B

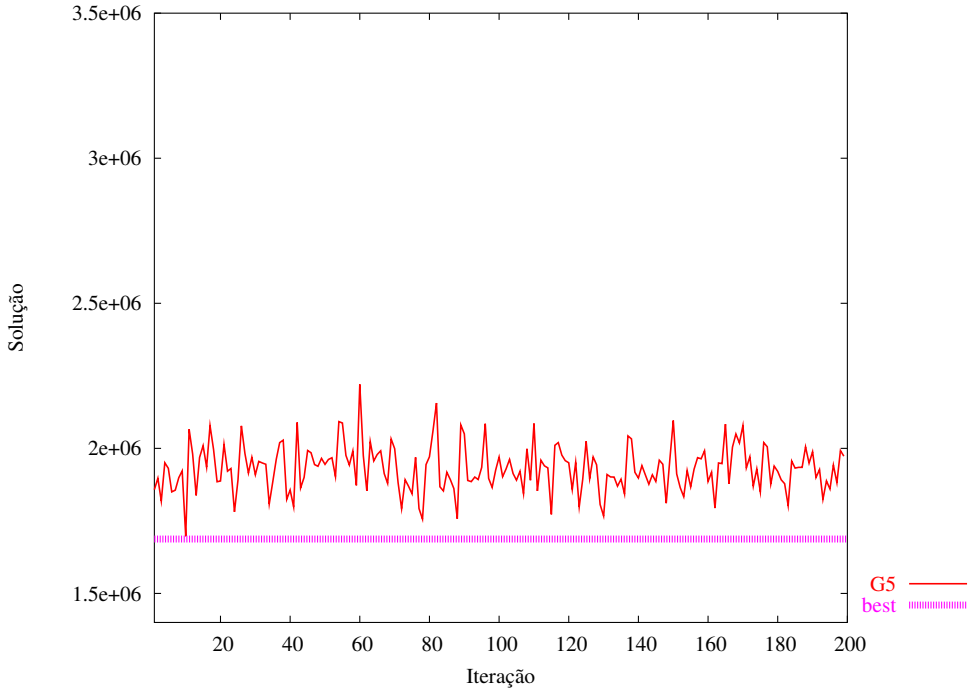

Figura 5.41: Solução por iteração - versão G5 - instância 100 do conjunto de testes B

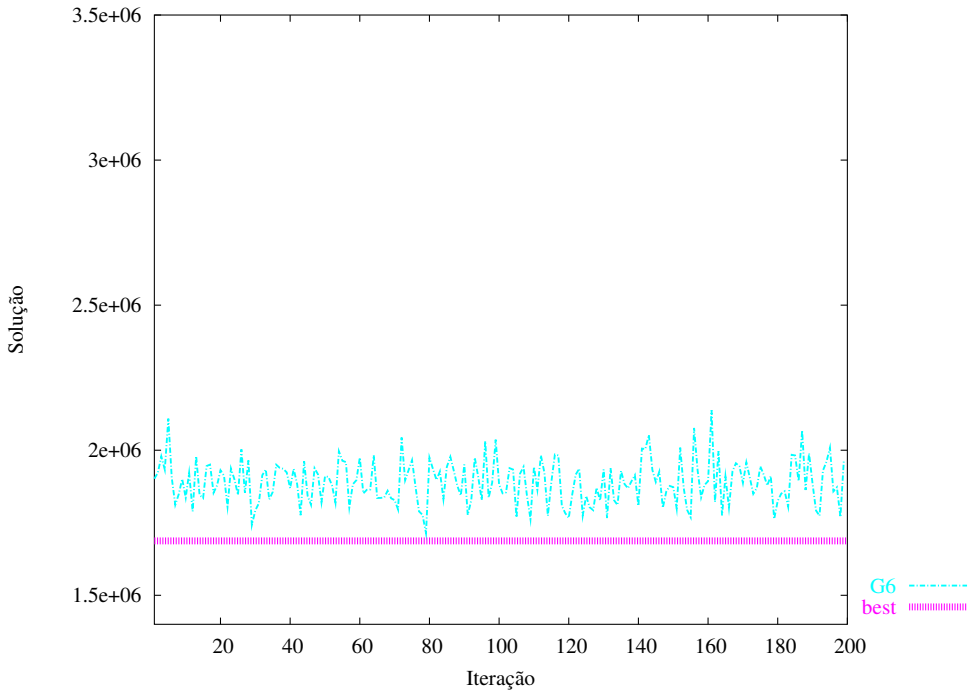

Figura 5.42: Solução por iteração - versão G6 - instância 100 do conjunto de testes B

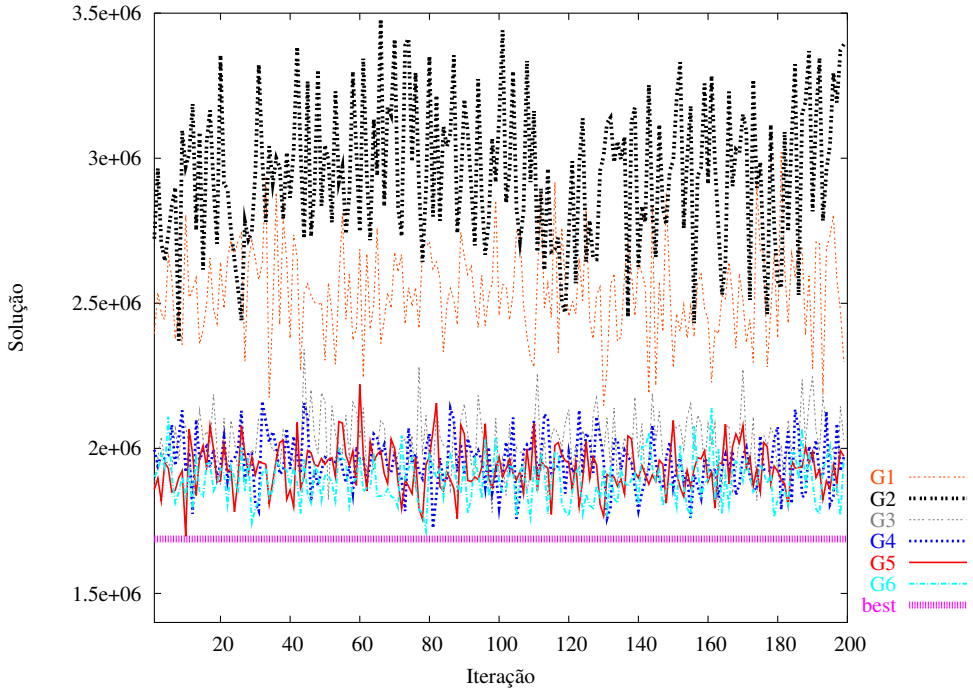

Figura 5.43: Solução por iteração das versões puras - instância 100 do conjunto de testes B

Os testes para a instância de 200 poços (vértices), para o conjunto de testes B, estão apresentados nas figuras 5.44 à 5.50.

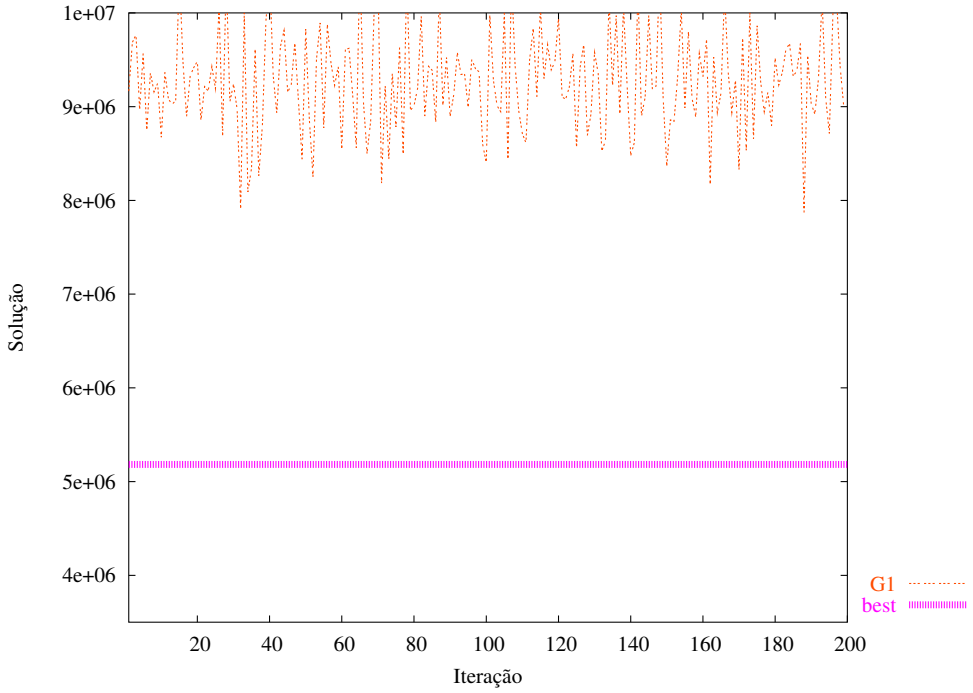

Figura 5.44: Solução por iteração - versão G1 - instância 200 do conjunto de testes B

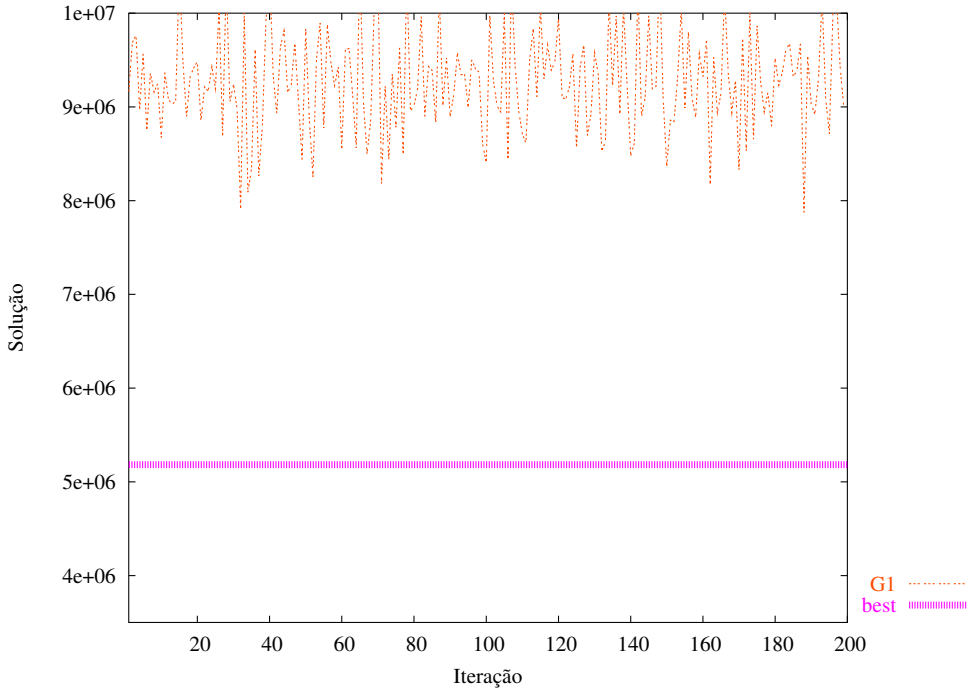

Figura 5.45: Solução por iteração - versão G2 - instância 200 do conjunto de testes B

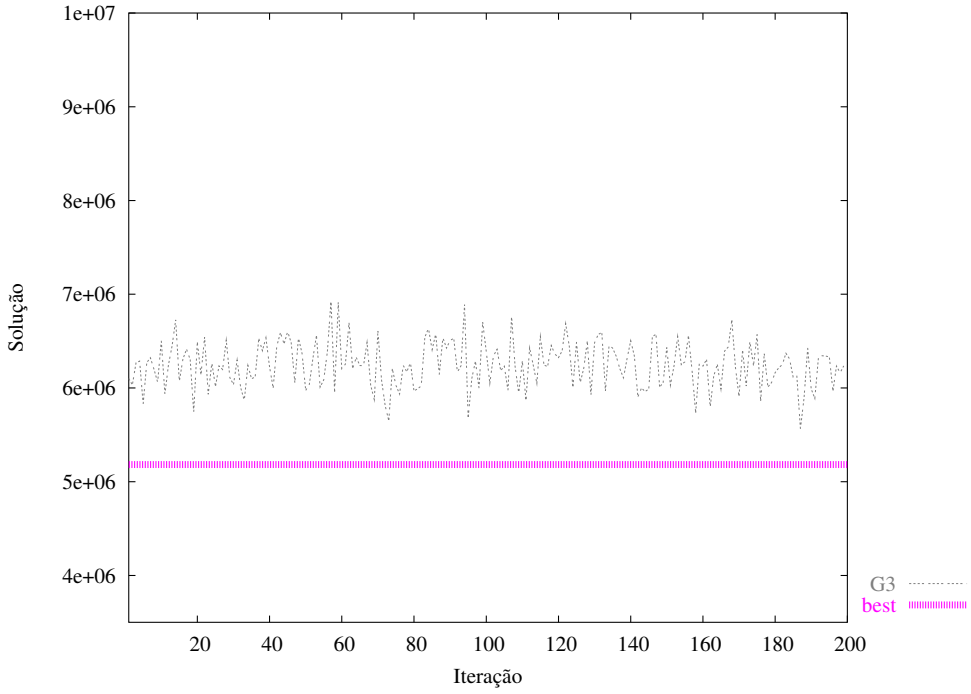

Figura 5.46: Solução por iteração - versão G3 - instância 200 do conjunto de testes B

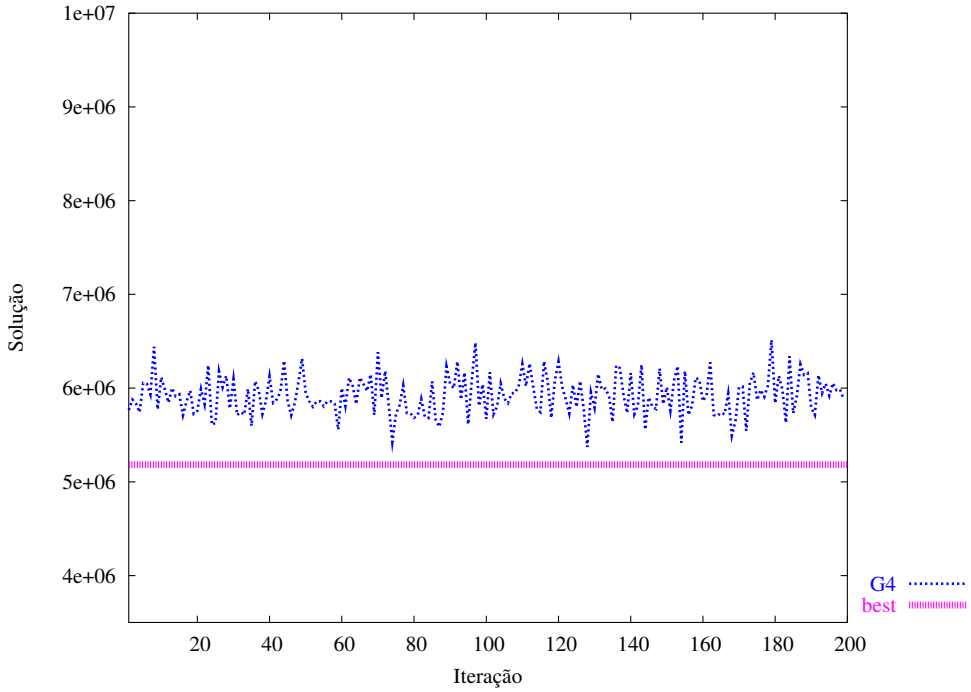

Figura 5.47: Solução por iteração - versão G4 - instância 200 do conjunto de testes B

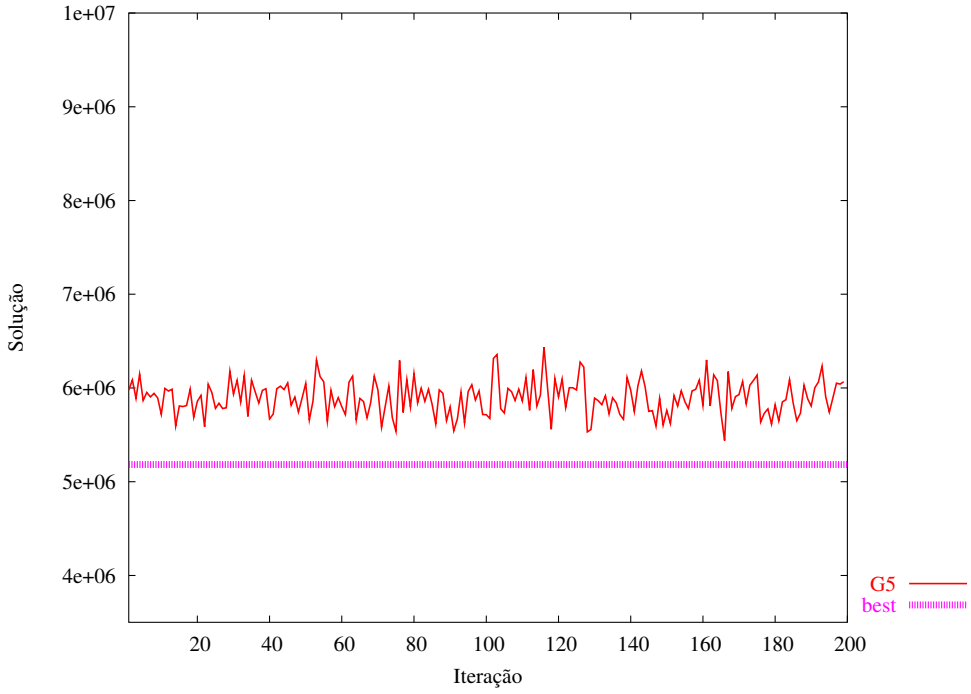

Figura 5.48: Solução por iteração - versão G5 - instância 200 do conjunto de testes B

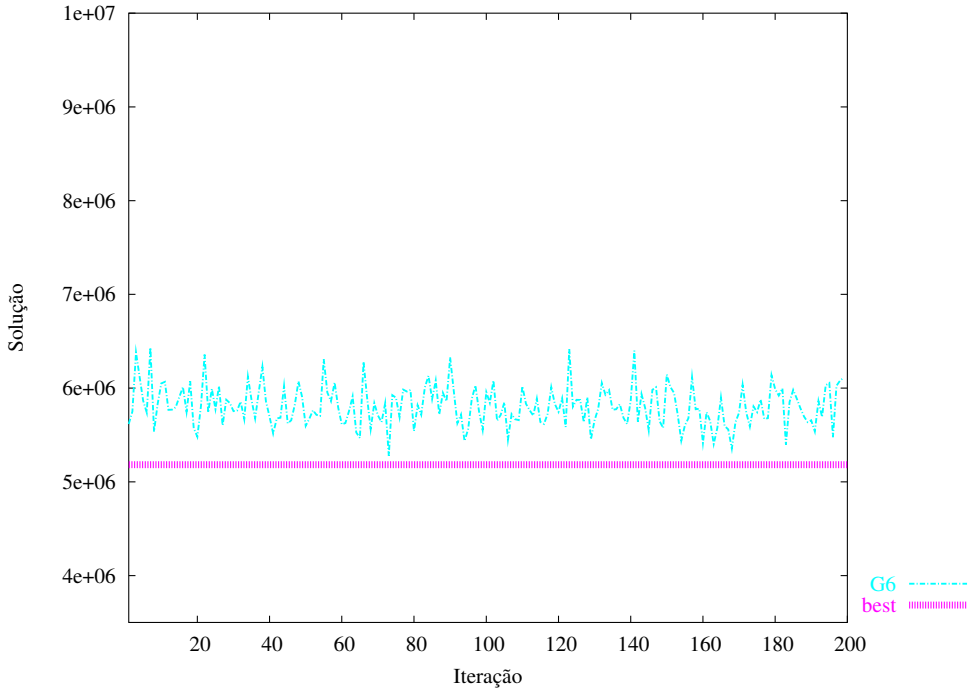

Figura 5.49: Solução por iteração - versão G6 - instância 200 do conjunto de testes B

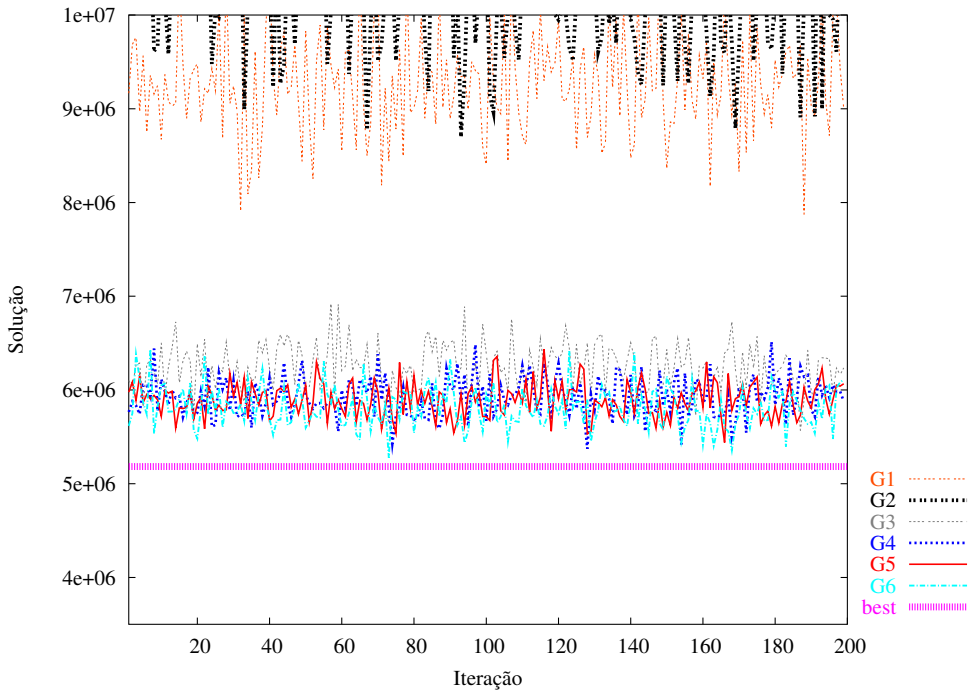

Figura 5.50: Solução por iteração das versões puras - instância 200 do conjunto de testes B

Após verificar os resultados para as versões puras (G1, G2, G3, G4, G5 e G6) para as instâncias de 50, 100 e 200 poços do conjunto de testes A e B podemos constatar que as versões se comportaram de forma similar para todos os casos, mostrando a superioridade das versões que incorporam a BL3 (G5 e G6) seguidas pelas versões que incorporam a BL2 (G3 e G4) e posteriormente pelas versões com a BL1 (G1 e G2).

A seguir realizamos os testes para as versões adaptativas usando as mesmas instâncias e os resultados são apresentados nas figuras de 5.51 à 5.74.

Nas figuras de 5.51 à 5.54 estão apresentadas as versões para a instâncias de 50 poços (vértices) do conjunto de testes A.

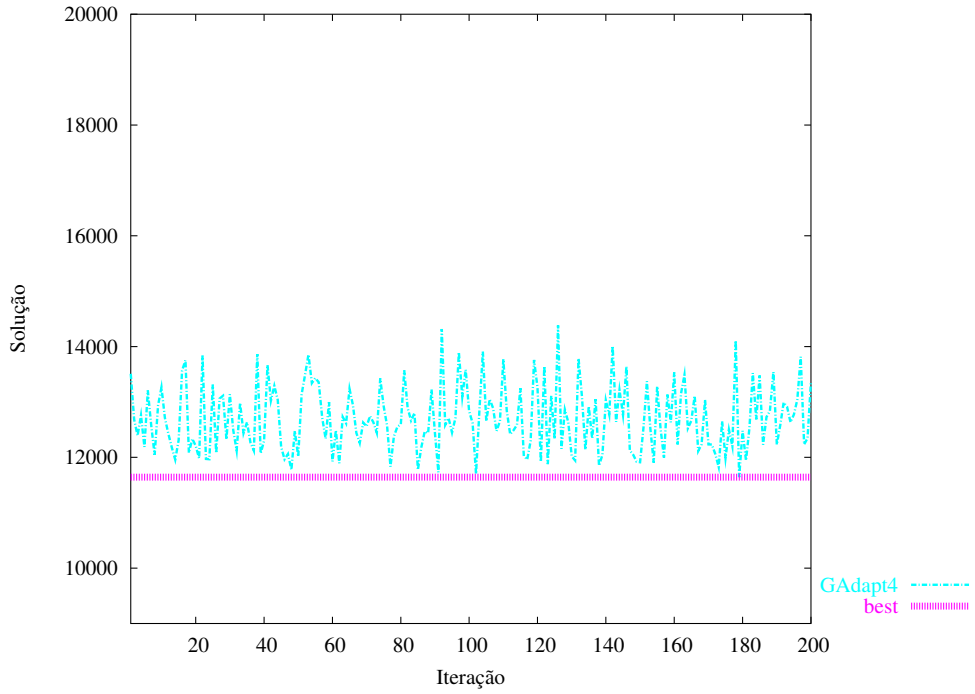

Figura 5.51: Solução por iteração - versão GAdapt4 - instância 50 do conjunto de testes A

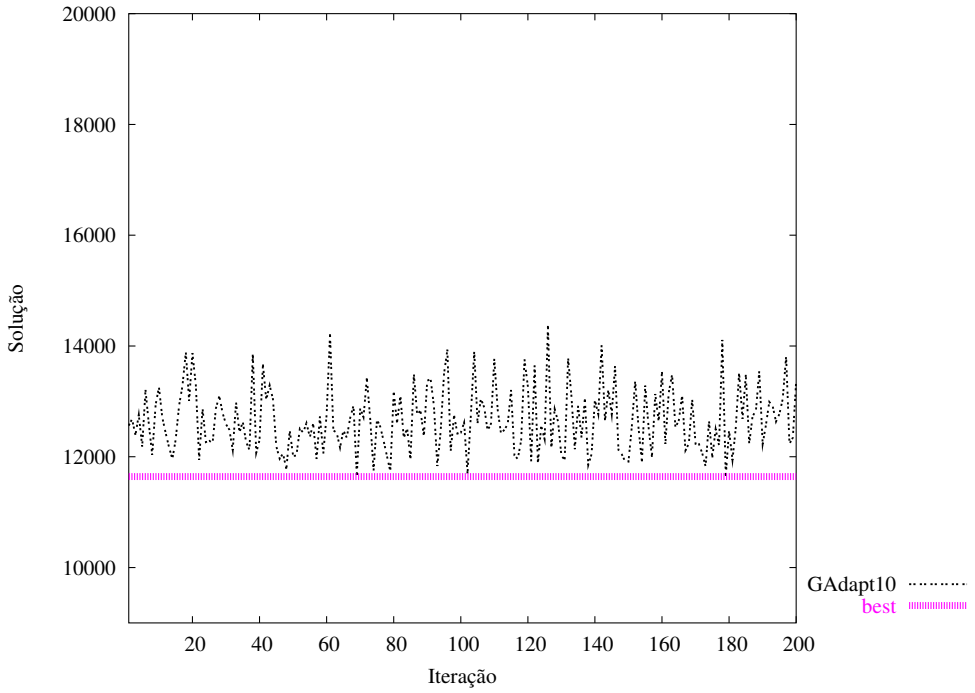

Figura 5.52: Solução por iteração - versão GAdapt10 - instância 50 do conjunto de testes A

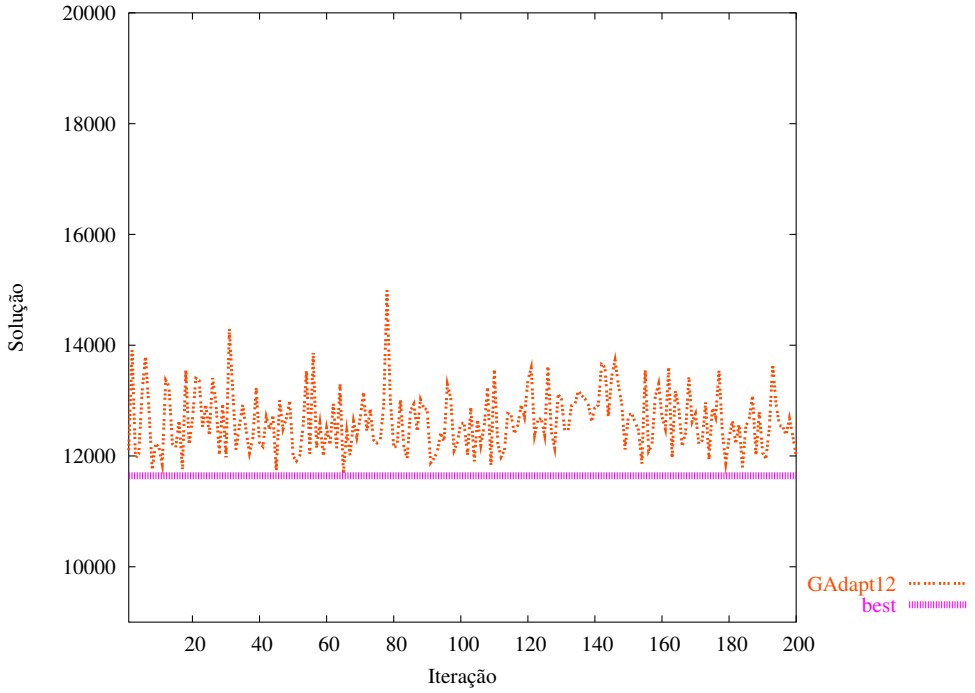

Figura 5.53: Solução por iteração - versão GAdapt12 - instância 50 do conjunto de testes A

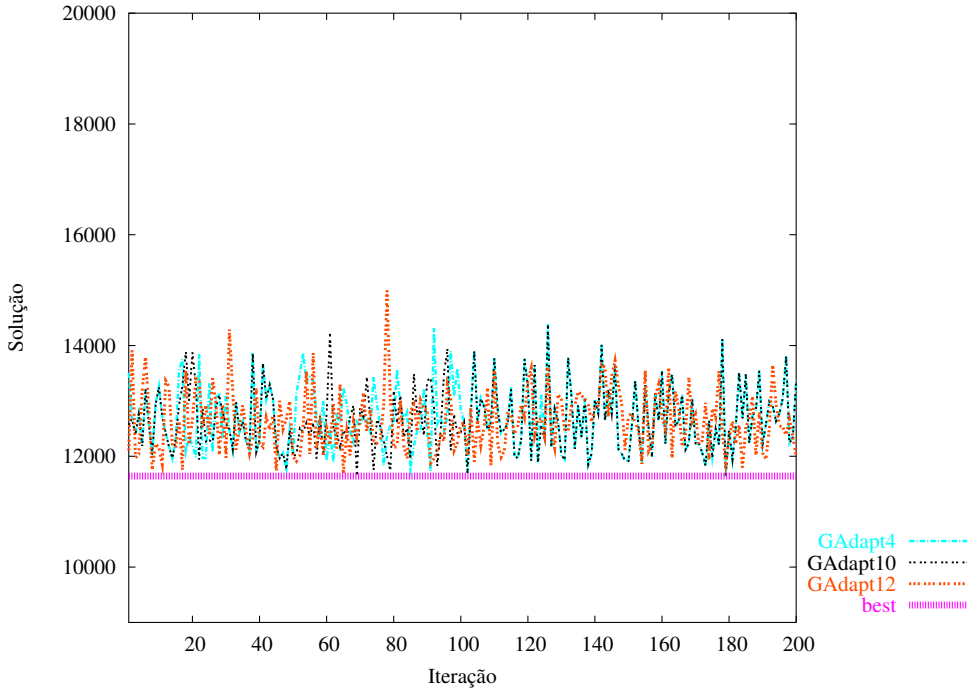

Figura 5.54: Solução por iteração - todas as versões adaptativas - instância 50 do conjunto de testes A

Nas figuras de 5.55 à 5.58 estão apresentadas as versões para a instâncias de 100 poços (vértices) do conjunto de testes A.

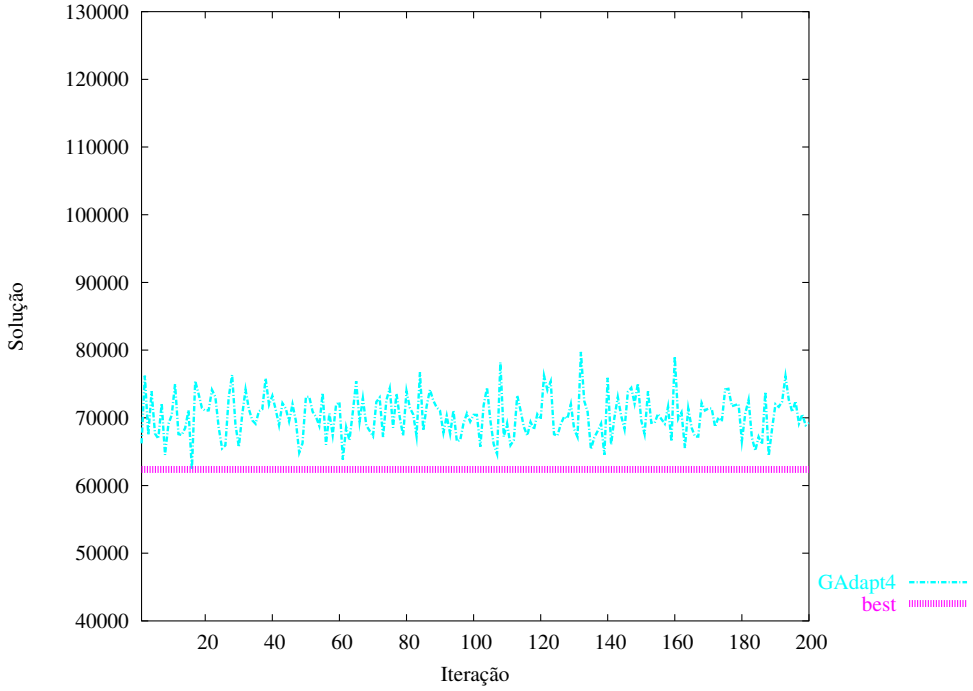

Figura 5.55: Solução por iteração - versão GAdapt4 - instância 100 do conjunto de testes A

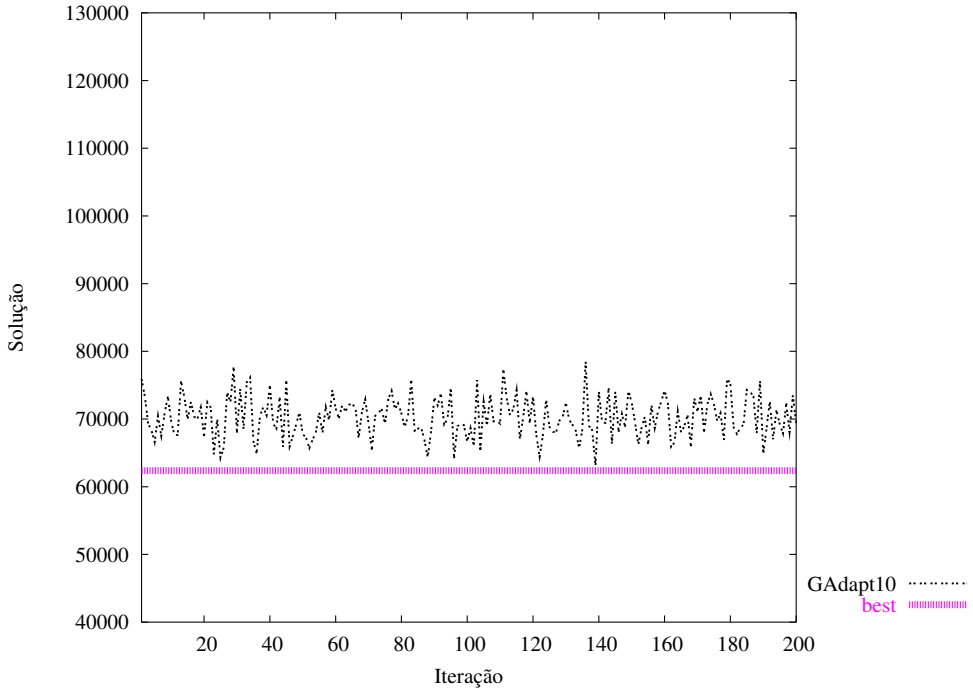

Figura 5.56: Solução por iteração - versão GAdapt10 - instância 100 do conjunto de testes A

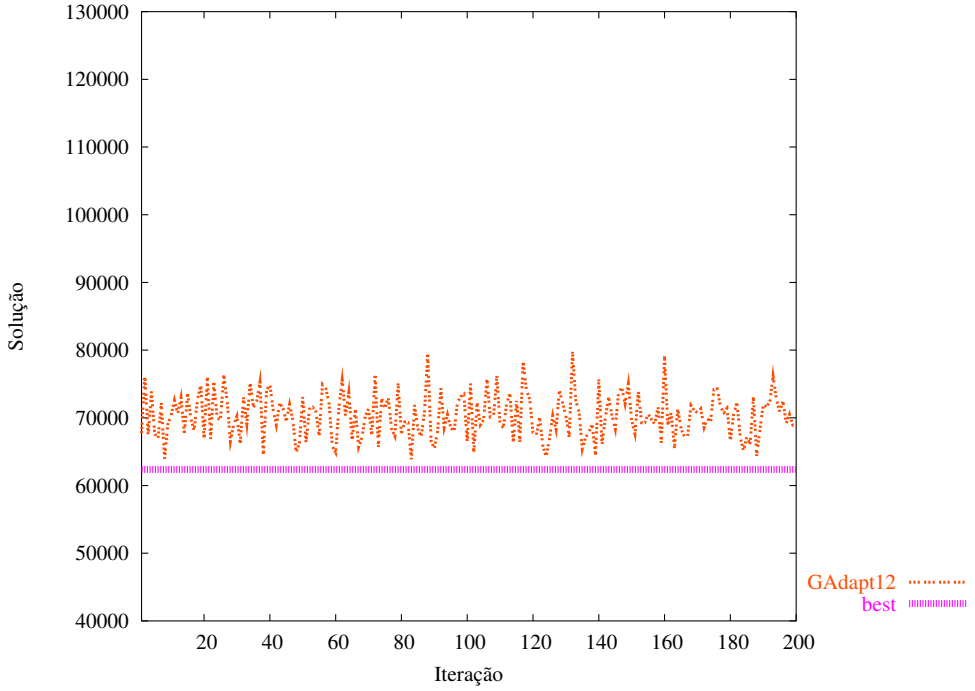

Figura 5.57: Solução por iteração - versão GAdapt12 - instância 100 do conjunto de testes A

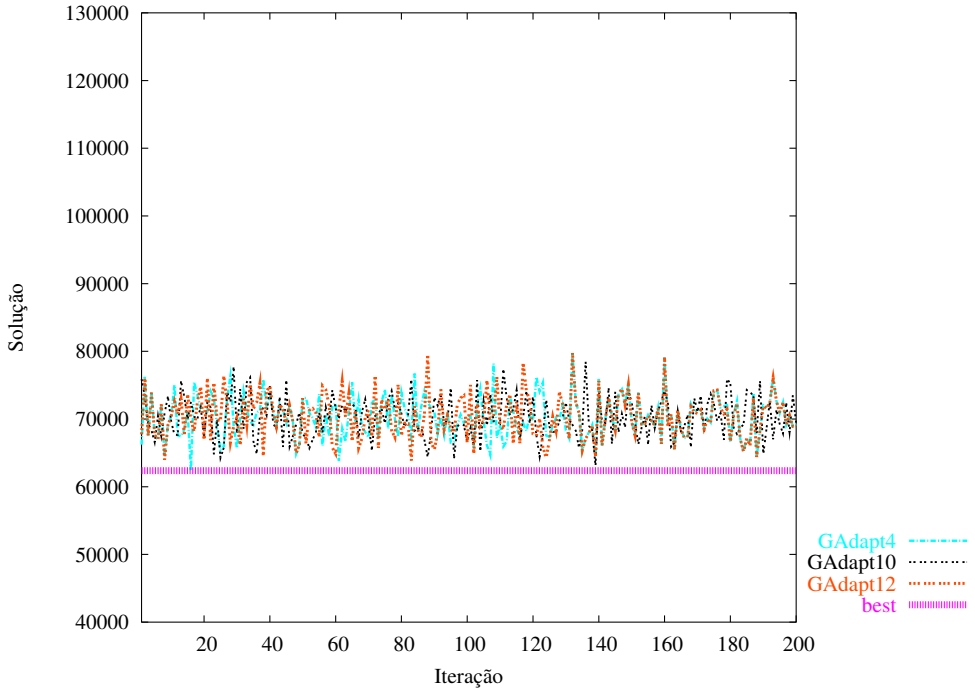

Figura 5.58: Solução por iteração - todas as versões adaptativas - instância 100 do conjunto de testes A

Nas figuras de 5.59 à 5.62 estão apresentadas as versões para a instâncias de 200 poços (vértices) do conjunto de testes A.

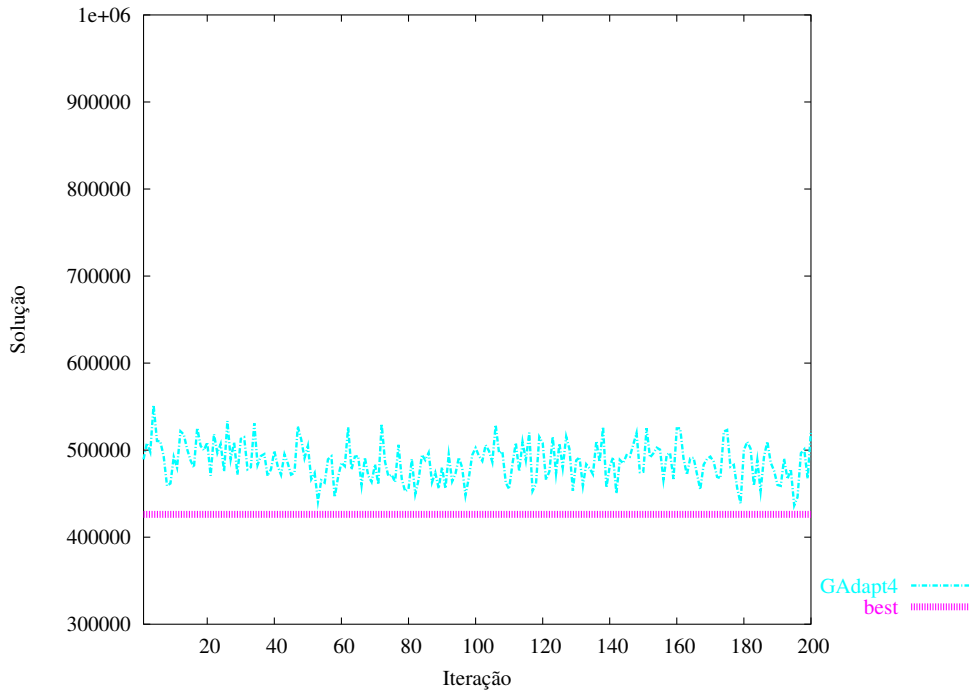

Figura 5.59: Solução por iteração - versão GAdapt4 - instância 200 do conjunto de testes A

Nas figuras de 5.63 à 5.66 estão apresentadas as versões para a instâncias de 50 poços (vértices) do conjunto de testes B.

Nas figuras de 5.67 à 5.70 estão apresentadas as versões para a instâncias de 100 poços (vértices) do conjunto de testes B.

Nas figuras de 5.71 à 5.74 estão apresentadas as versões para a instâncias de 200 poços (vértices) do conjunto de testes B.

Os resultados das versões adaptativas, ilustradas pelas figuras 5.51 à 5.74, comprovam que a meta inicialmente colocada foi atendida, ao menos nestes testes empíricos. Ou seja, obter versões GRASP onde o resultado de cada iteração tende a sofrer alguma influência do resultado das iterações anteriores.

Ao se introduzir uma fase de aprendizado (memória) no GRASP, como é o caso das versões com Reconexão por Caminhos G7, G8 e G9 e as versões adaptativas, de certa forma passa-se a usar critérios elitistas, como a criação do conjunto elite ou a calibração

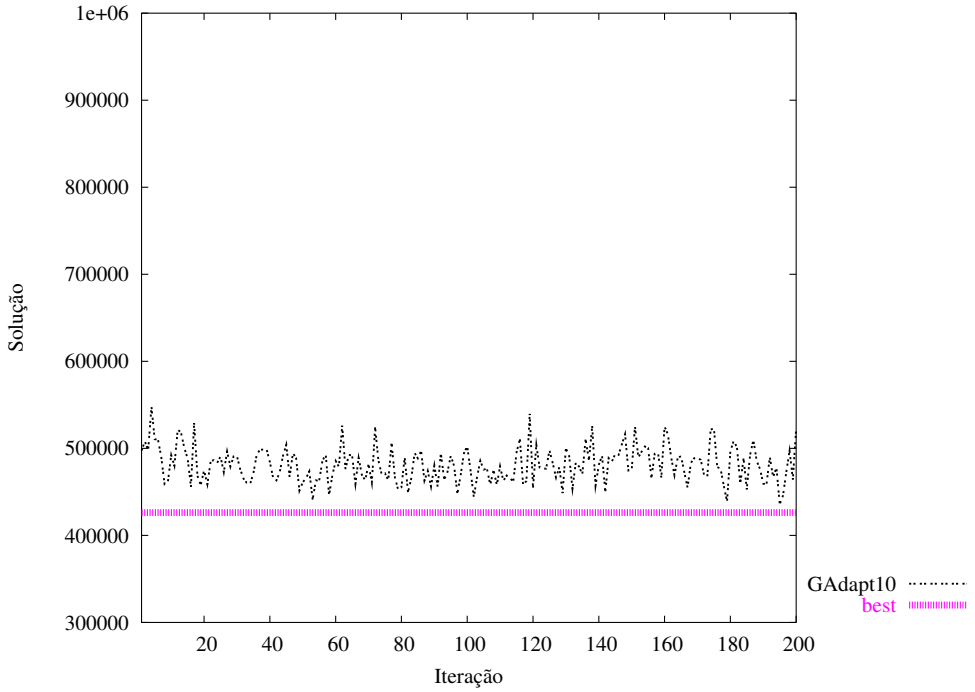

Figura 5.60: Solução por iteração - versão GAdapt10 - instância 200 do conjunto de testes A

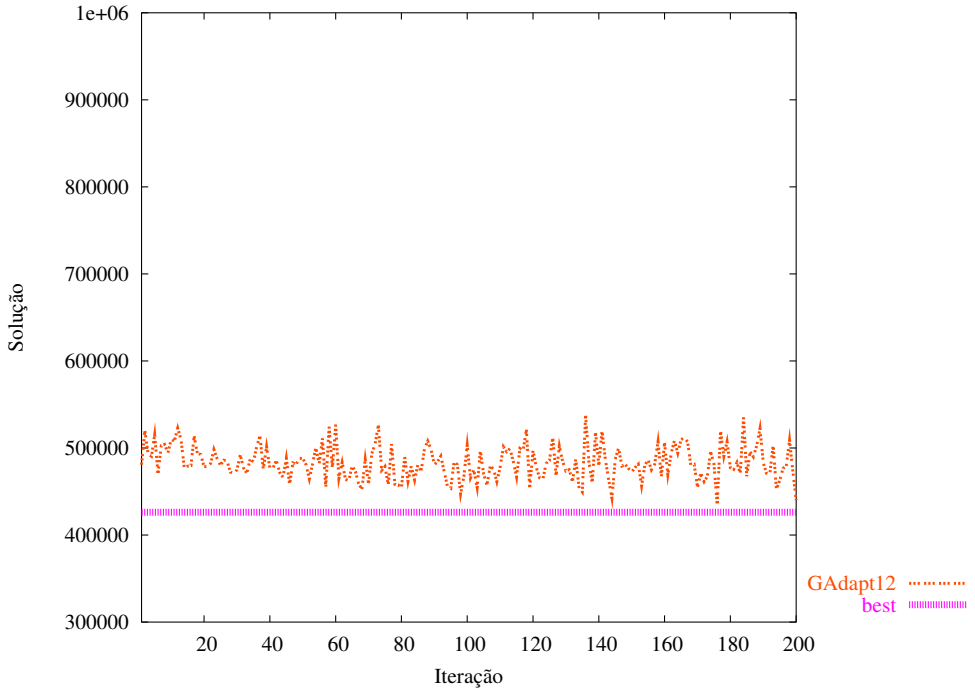

Figura 5.61: Solução por iteração - versão GAdapt12 - instância 200 do conjunto de testes A

de parâmetros nas versões adaptativas.

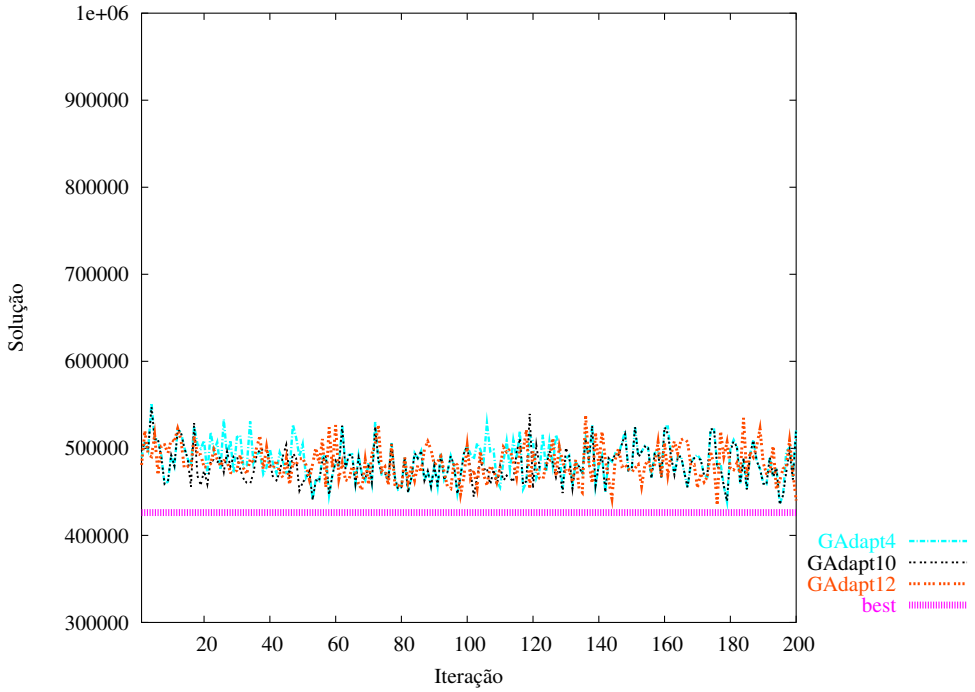

Figura 5.62: Solução por iteração - todas as versões adaptativas - instância 200 do conjunto de testes A

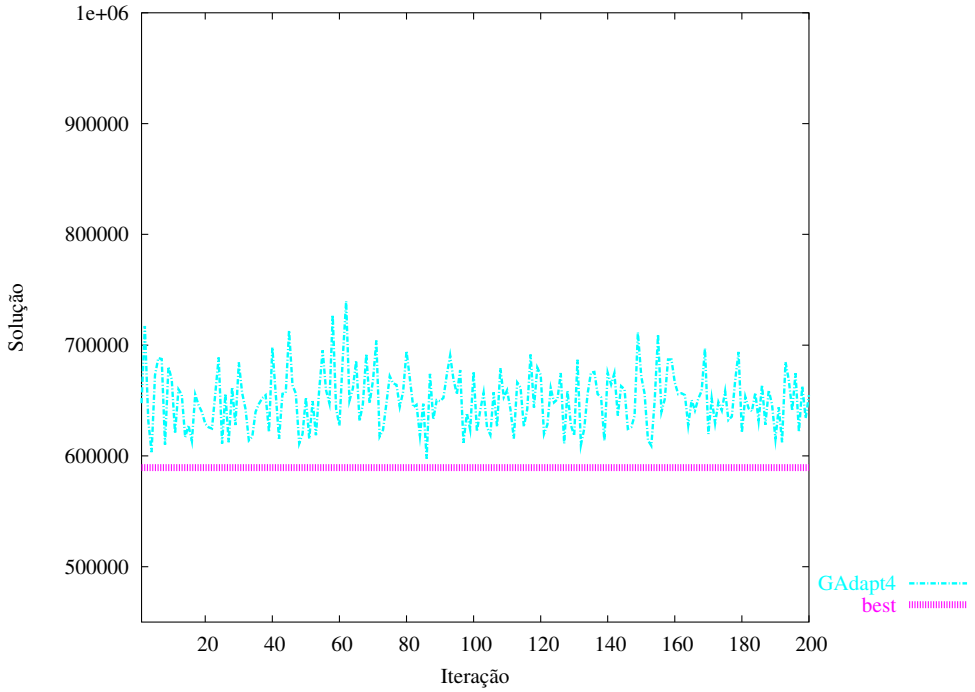

Figura 5.63: Solução por iteração - versão GAdapt4 - instância 50 do conjunto de testes B

Com isso, observamos um comportamento mais regular, tanto nas soluções por iteração (figuras 5.51 à 5.74) como uma regularidade nas diferentes execuções do GRASP

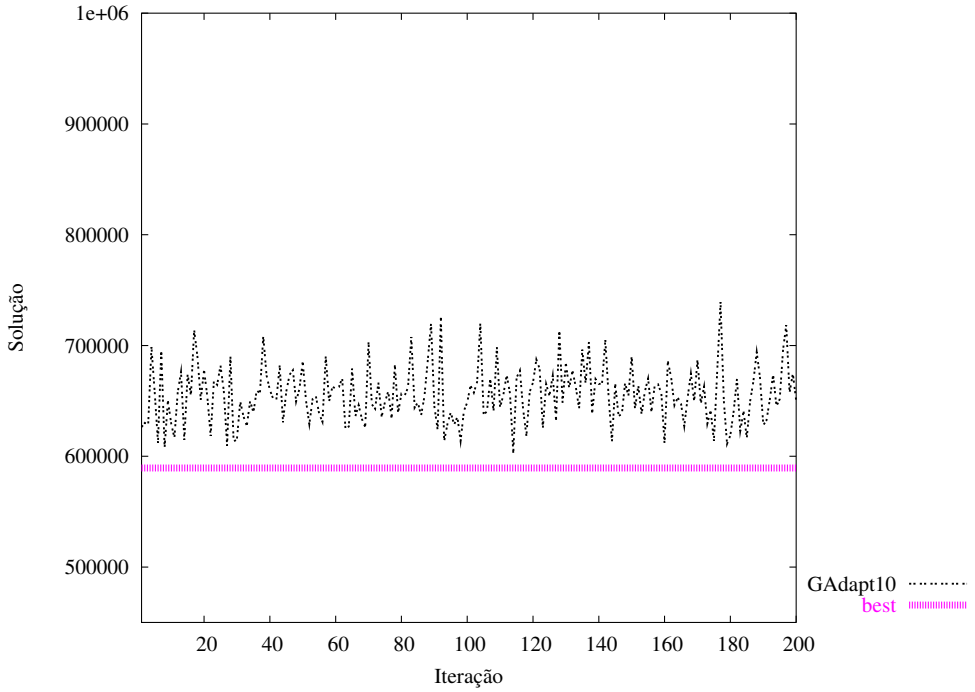

Figura 5.64: Solução por iteração - versão GAdapt10 - instância 50 do conjunto de testes B

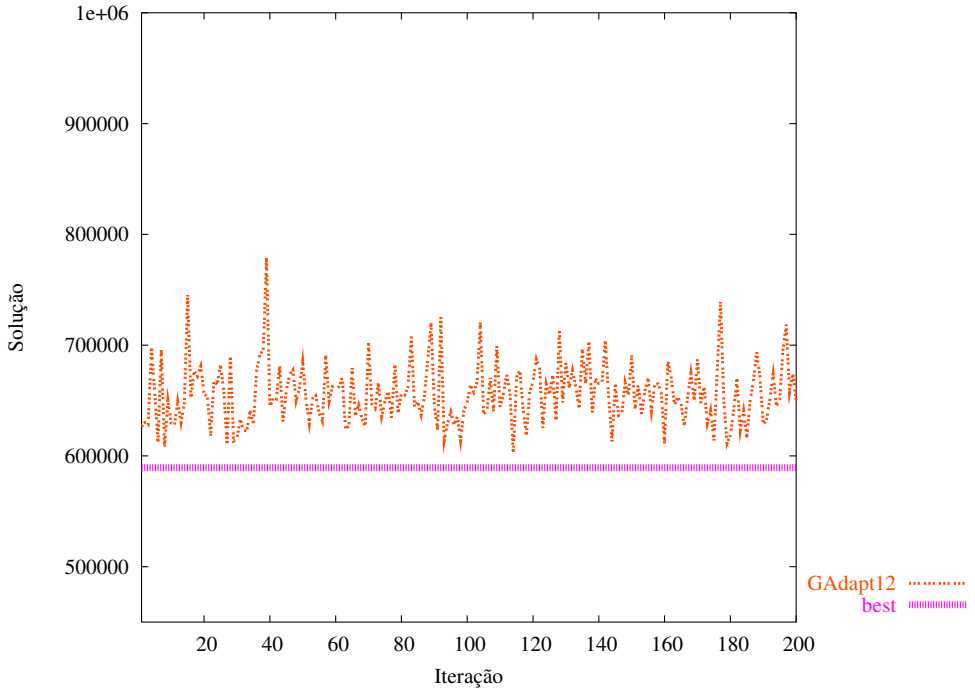

Figura 5.65: Solução por iteração - versão GAdapt12 - instância 50 do conjunto de testes B

(análise probabilística empírica). Com isso, acreditamos ter alcançado para estas versões uma regularidade que as torna mais confiáveis para diferentes aplicações reais.

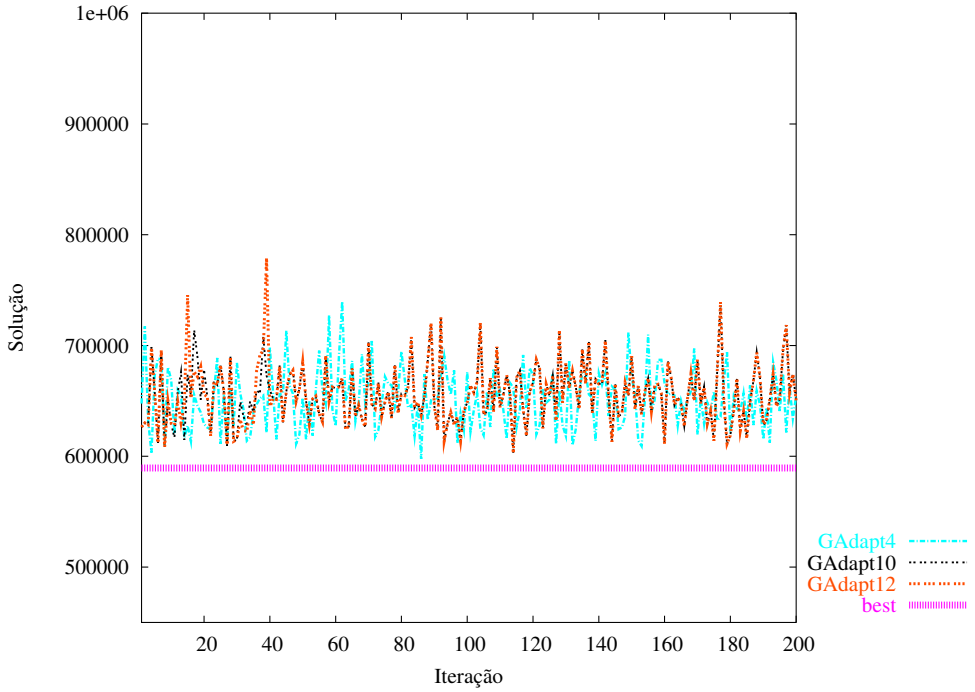

Figura 5.66: Solução por iteração - todas as versões adaptativas - instância 50 do conjunto de testes B

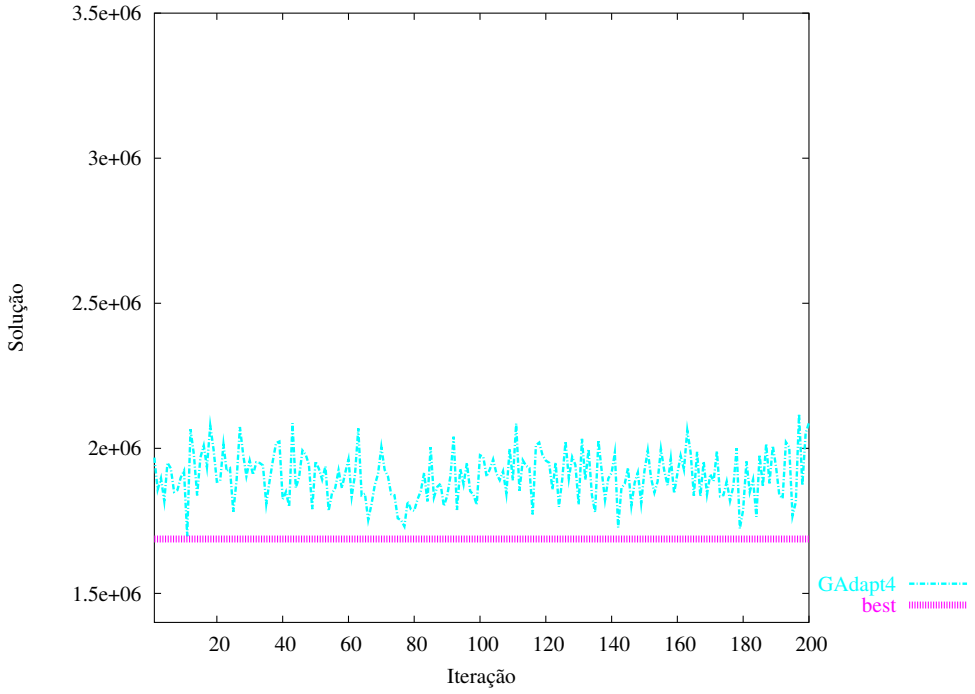

Figura 5.67: Solução por iteração - versão GAdapt4 - instância 100 do conjunto de testes B

5.9 Tempos Computacionais

Nesta seção apresentamos uma outra análise dos tempos computacionais consumidos pelas versões para executar as 200 iterações GRASP (parâmetro de finalização utilizado pelas

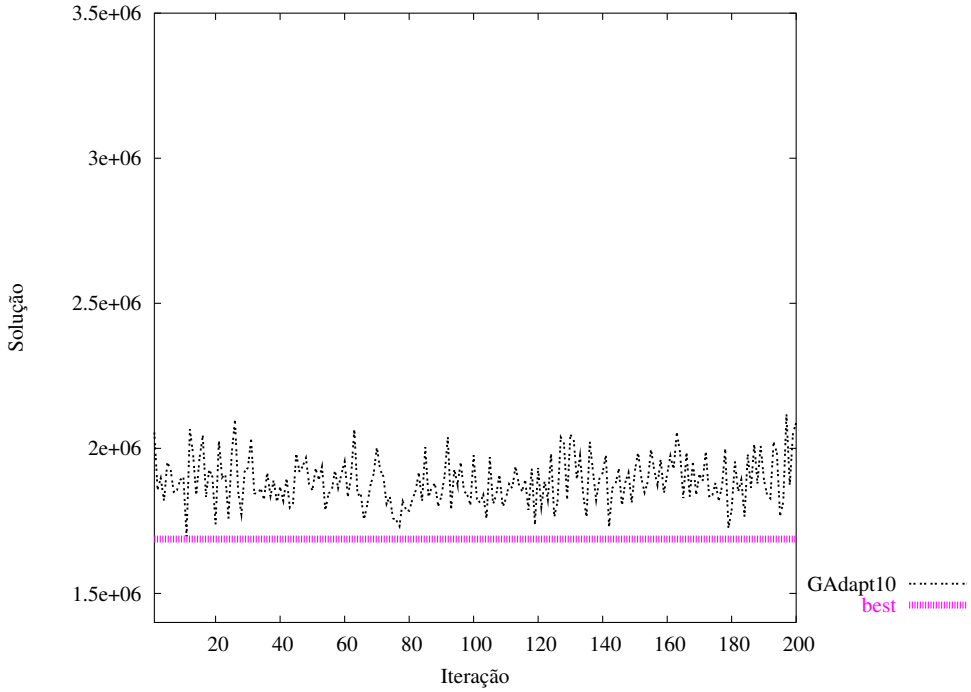

Figura 5.68: Solução por iteração - versão GAdapt10 - instância 100 do conjunto de testes B

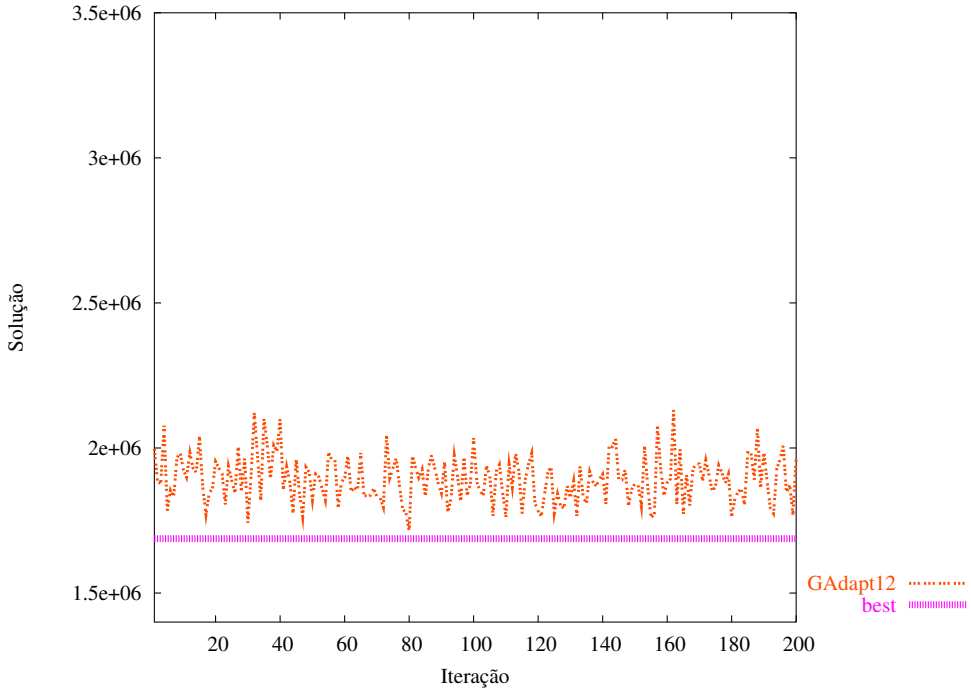

Figura 5.69: Solução por iteração - versão GAdapt12 - instância 100 do conjunto de testes B

versões GRASP aqui propostas).

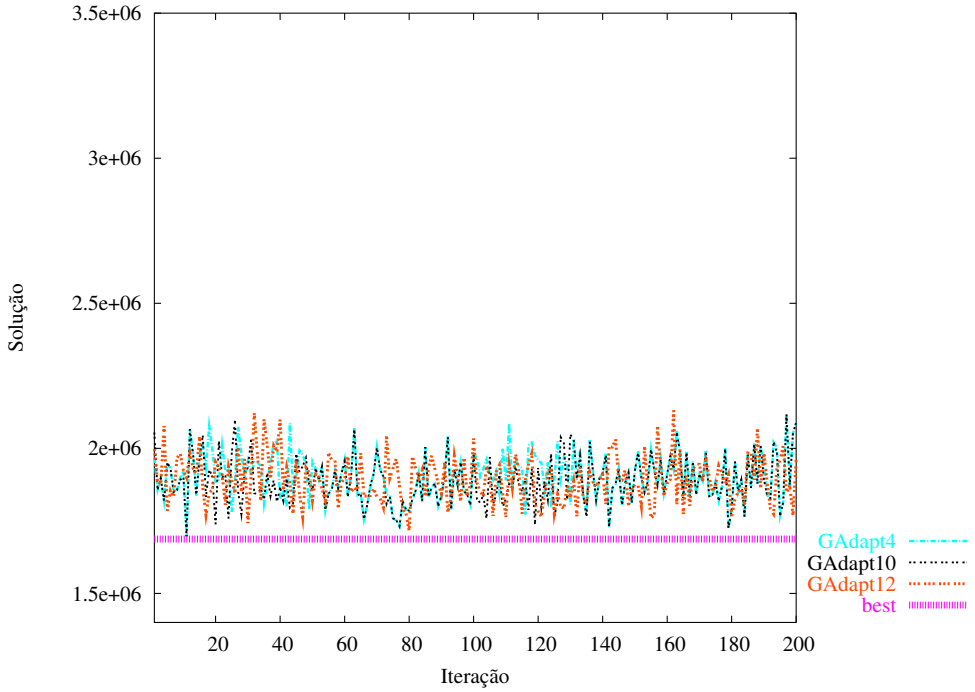

Figura 5.70: Solução por iteração - todas as versões adaptativas - instância 100 do conjunto de testes B

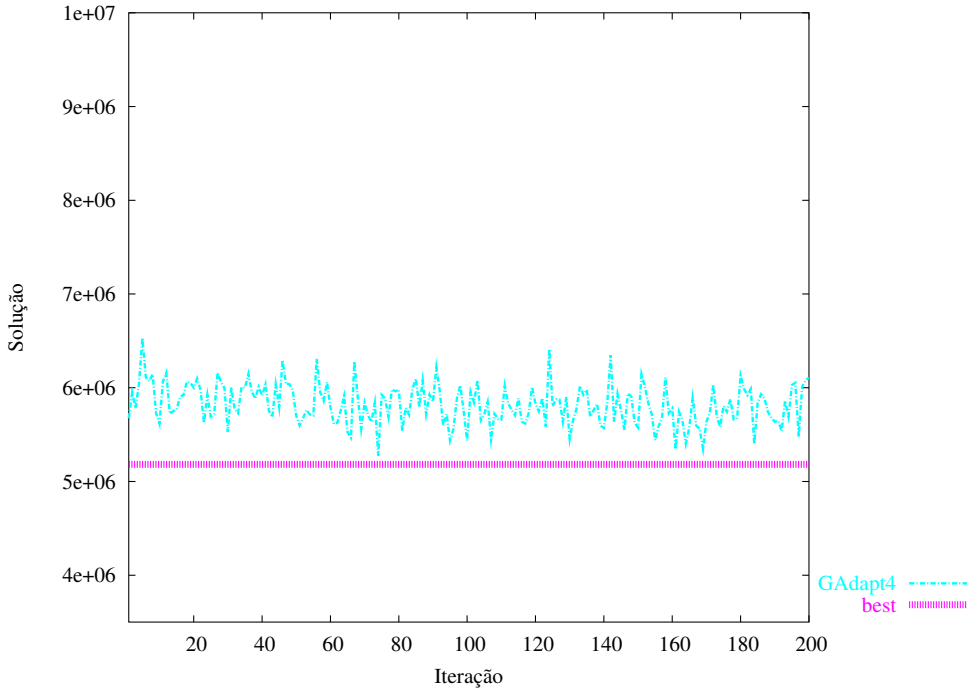

Figura 5.71: Solução por iteração - versão GAdapt4 - instância 200 do conjunto de testes B

Os gráficos das versões puras estão apresentados das figuras 5.75 à 5.79, onde, no eixo X, o valor 50 representa a instância de 50 poços, 100 a de 100 poços e assim por diante.

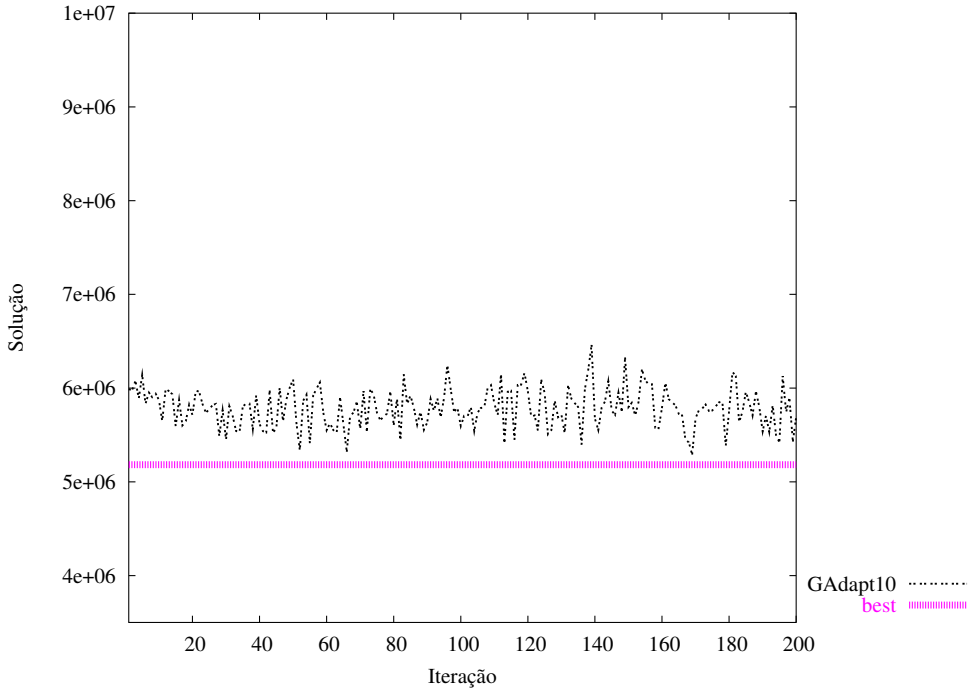

Figura 5.72: Solução por iteração - versão GAdapt10 - instância 200 do conjunto de testes B

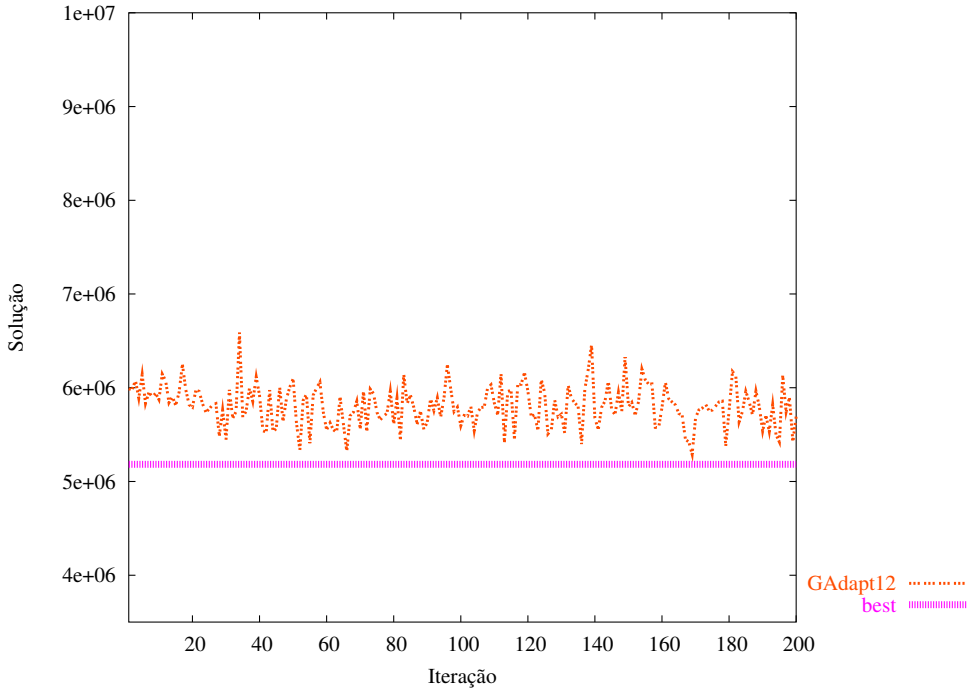

Figura 5.73: Solução por iteração - versão GAdapt12 - instância 200 do conjunto de testes B

Todas as instâncias foram consideradas nesta análise, ou seja, 50, 100, 200, 300, 400, 500, 700 e 1000 poços (vértices).

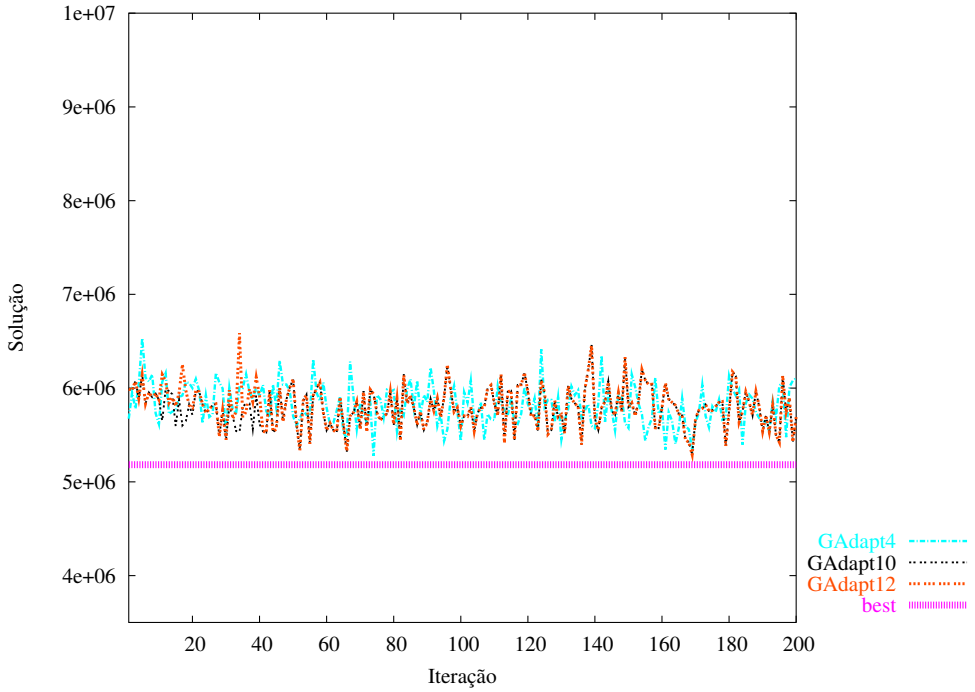

Figura 5.74: Solução por iteração - todas as versões adaptativas - instância 200 do conjunto de testes B

Nas figuras 5.75 e 5.76 são apresentados os tempos computacionais das versões puras (G1 à G6) para as instâncias do conjunto de testes A.

Podemos concluir pelas figuras 5.75 e 5.76 que o tempo computacional consumido pelo algoritmo de construção é muito pequeno em relação ao total de tempo gasto pelo GRASP, já que as versões que utilizam o mesmo construtivo, porém buscas locais diferentes, possuem uma diferença de tempo computacional muito acentuada. Ou seja, os algoritmos que utilizam a busca local BL3 (G5 e G6) consomem muito mais tempo computacional do que os algoritmos que utilizam a BL2 (G3 e G4) e a BL1 respectivamente (G1 e G2), utilizando como critério de parada um número fixo de iterações.

Nas figuras 5.77 e 5.78 são apresentados os tempos computacionais das versões puras (G1 à G6) para as instâncias do conjunto de testes B.

Os resultados para as instâncias do conjunto de testes B, observados nas figuras 5.77 e 5.78 não apresentaram diferenças bruscas quantos aos resultados observados para o conjunto de testes A.

Nas figuras 5.79 e 5.81 são feitas comparações dos tempos computacionais da versão

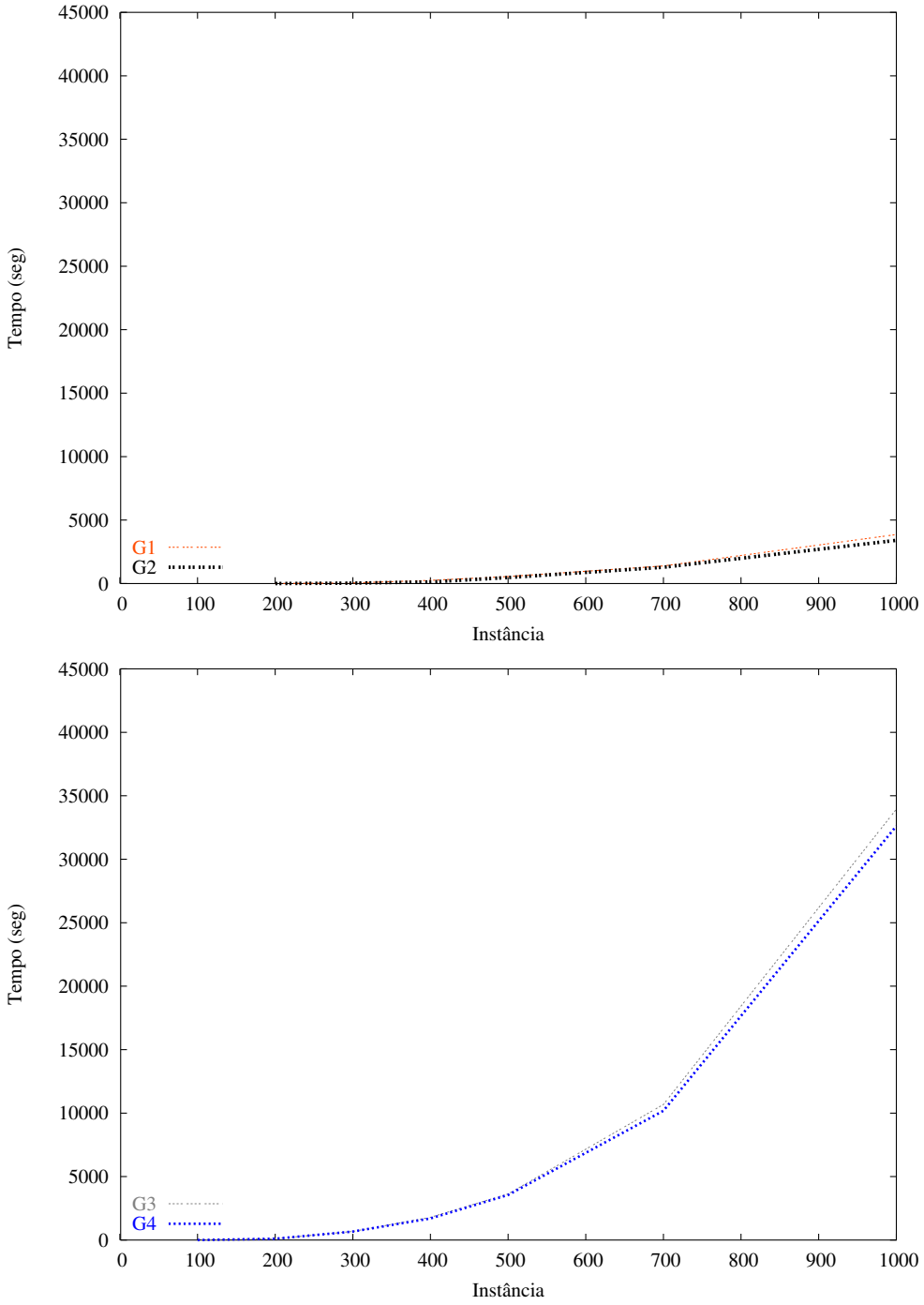

Figura 5.75: Tempo Computacional - versões G1 e G2 e versões G3 e G4 do Conjunto de testes A

G6 com as versões que incorporam o módulo de RC (G7 e G8), para verificar o aumento do tempo computacional ao incluir um módulo de RC em uma versão pura.

Lembramos que a versão G8 é a versão G6 com a execução do módulo de RC apenas no final de toda a execução da heurística GRASP. Portanto a diferença entre as curvas

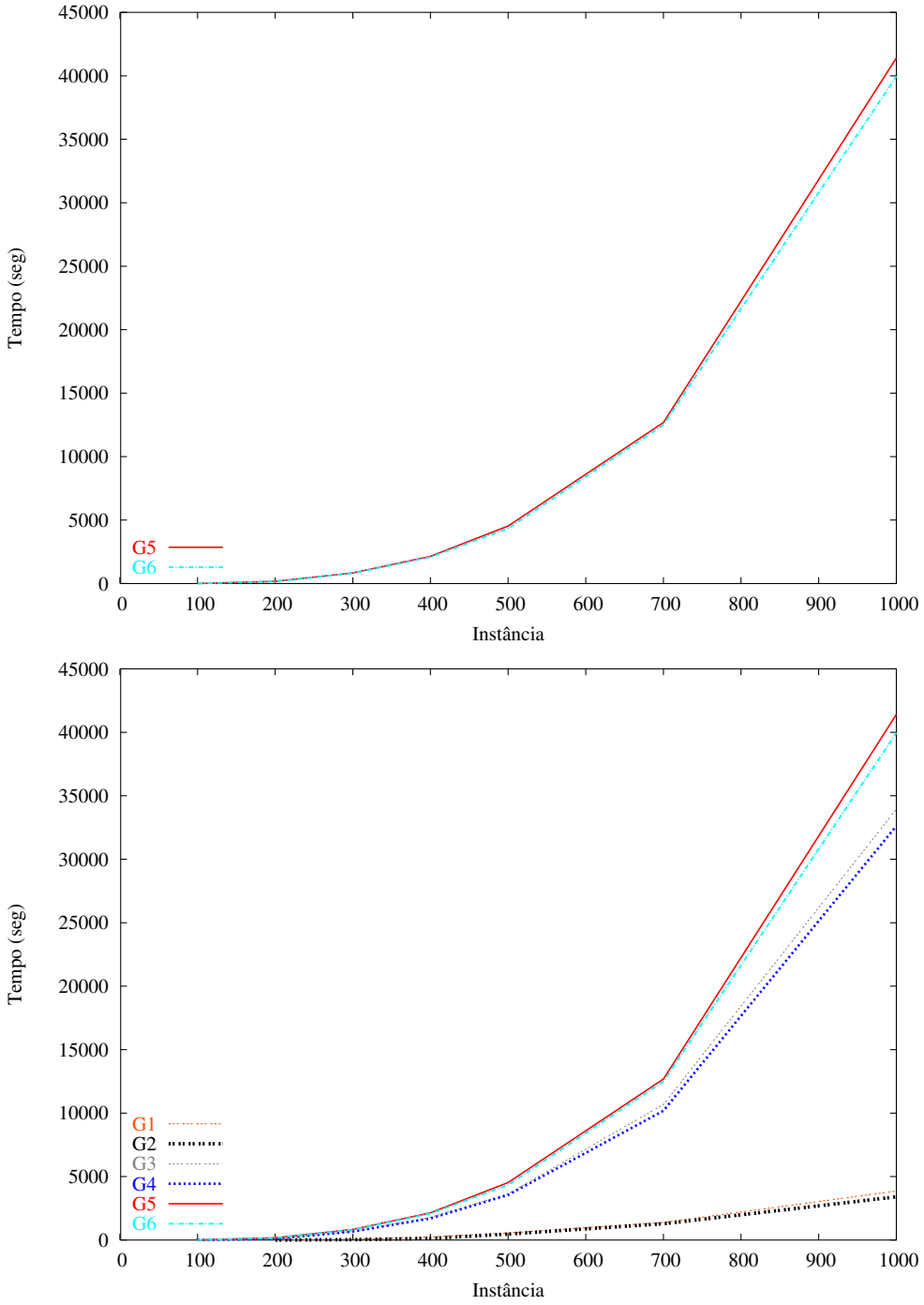

Figura 5.76: Tempo Computacional - versões G5 e G6 e no segundo gráfico todas as versões de G1 à G6 do Conjunto de testes A

de G6 e G8 se refere a uma execução de RC.

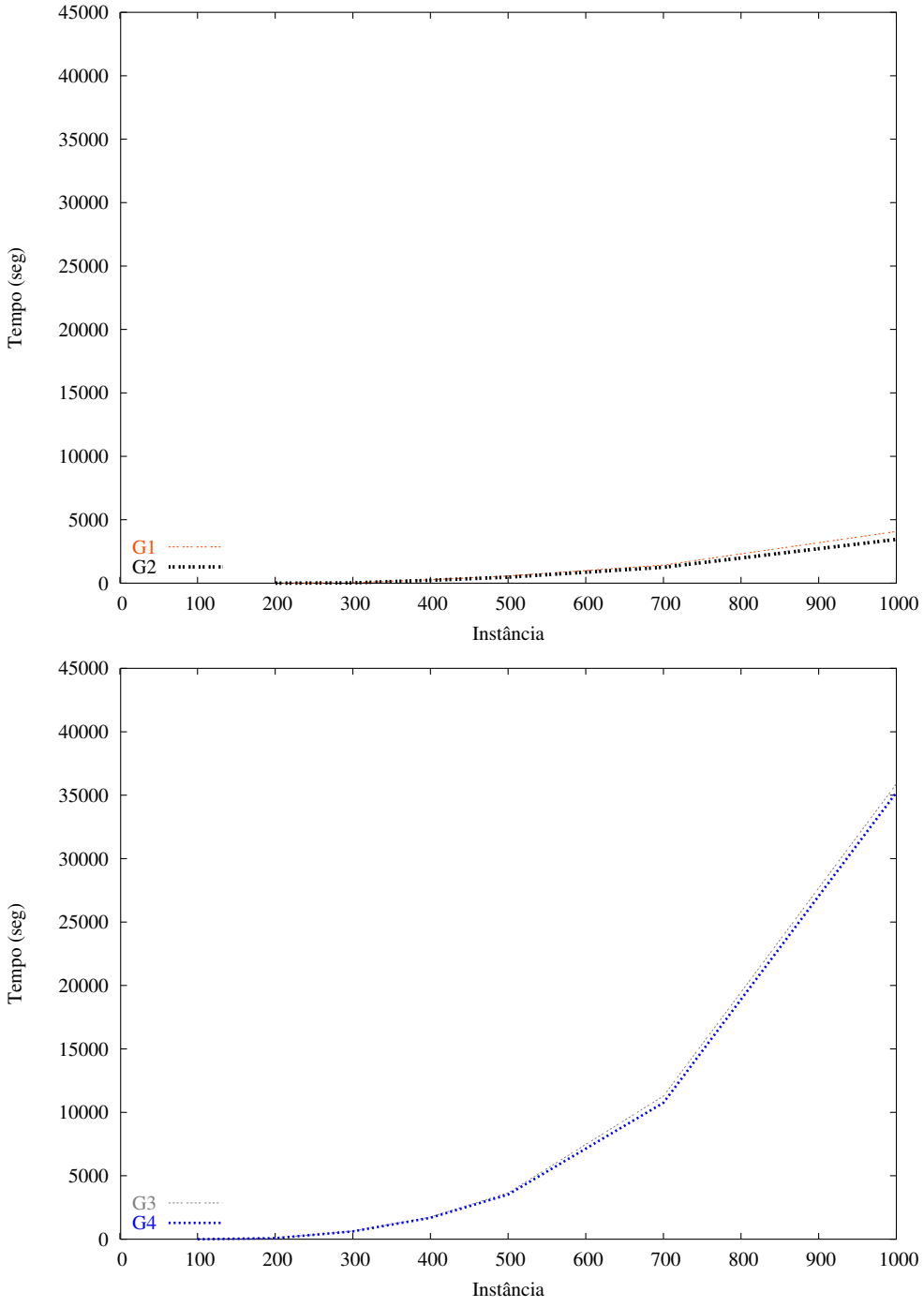

Figura 5.77: Tempo Computacional - versões G1 e G2 e versões G3 e G4 do Conjunto de testes B

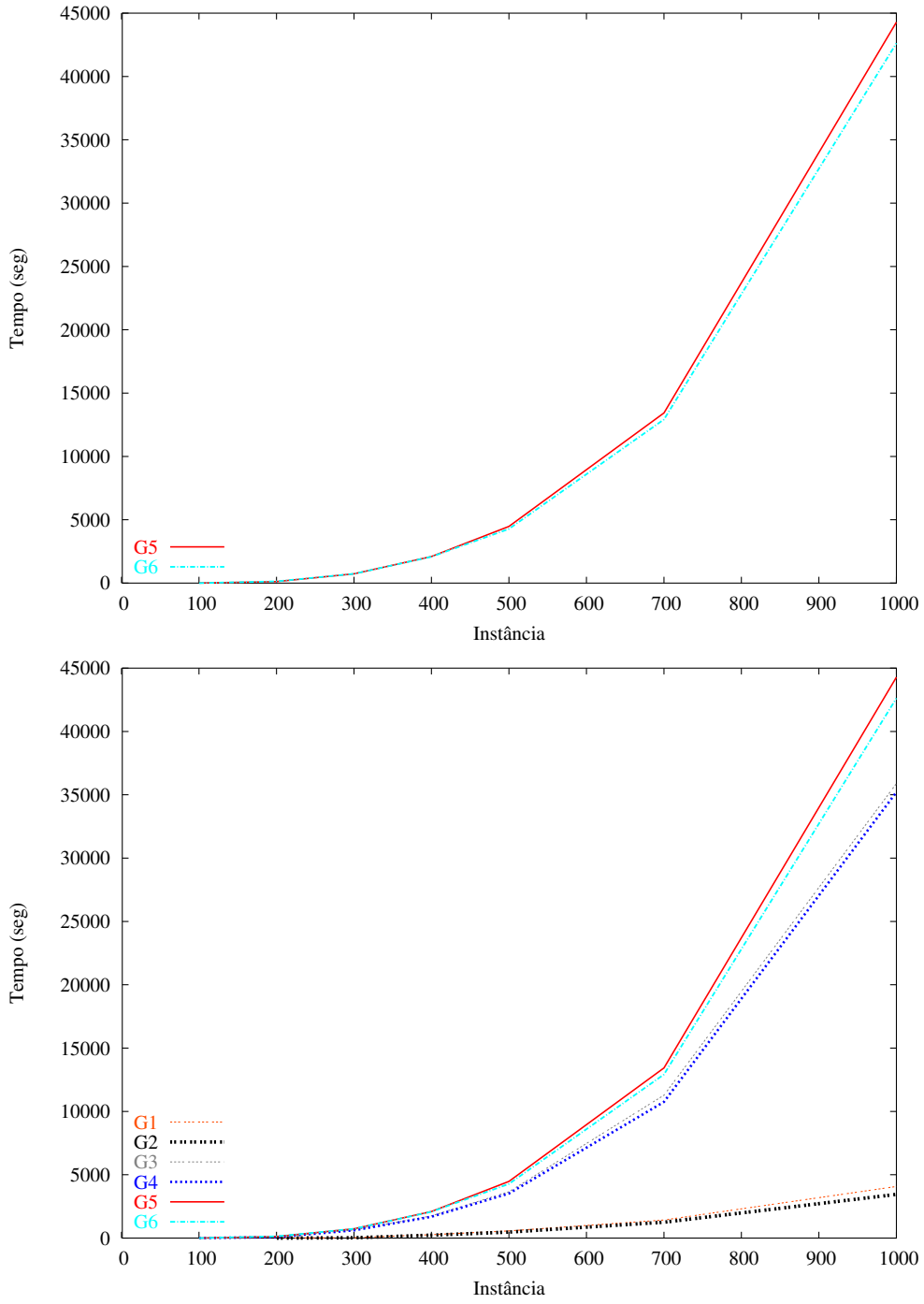

Figura 5.78: Tempo Computacional - versões G5 e G6 e no segundo gráfico todas as versões de G1 à G6 do Conjunto de testes B

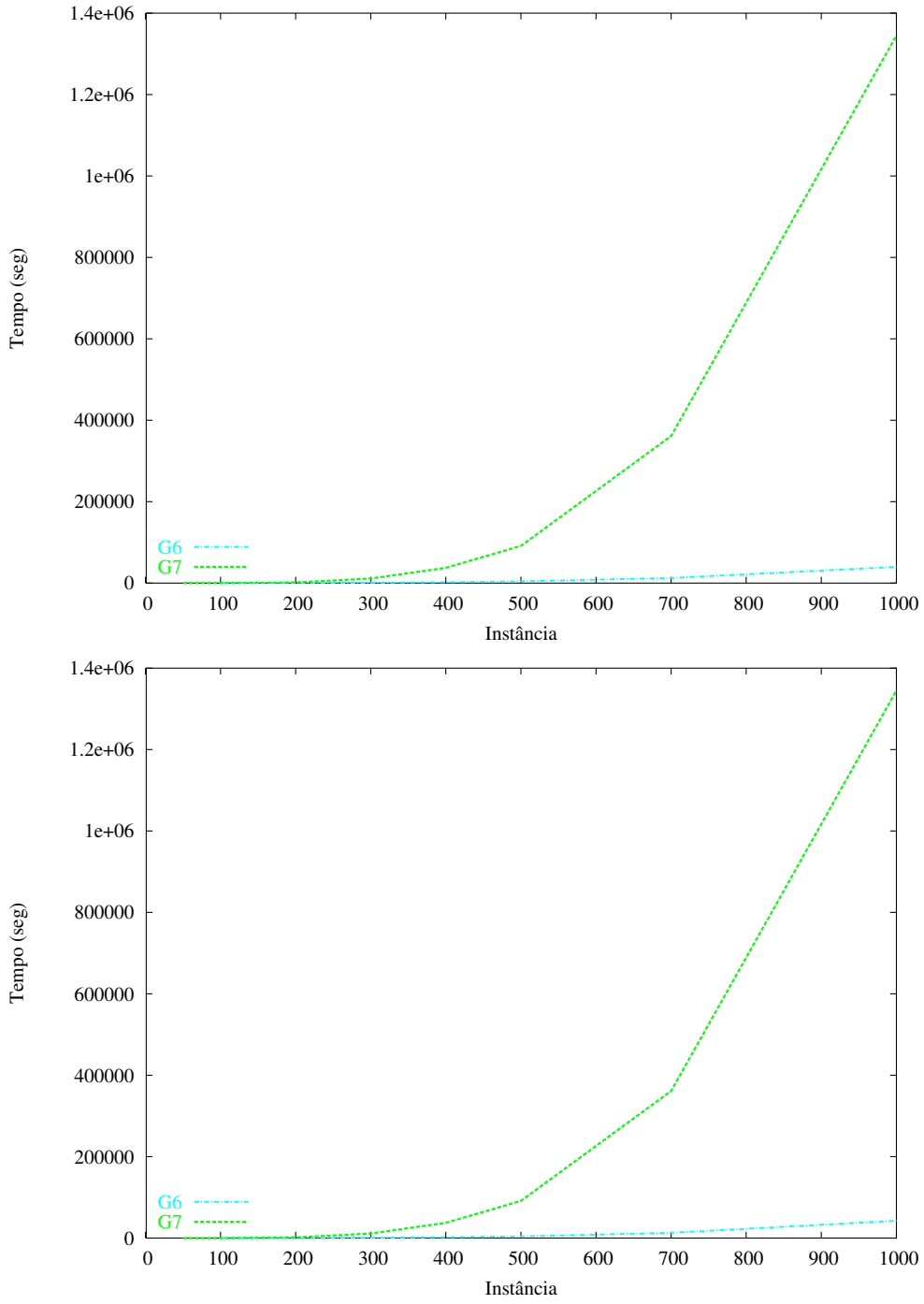

Figura 5.79: Tempo Computacional - versões G6 e G7 para as instâncias do Conjunto de testes A e B, respectivamente

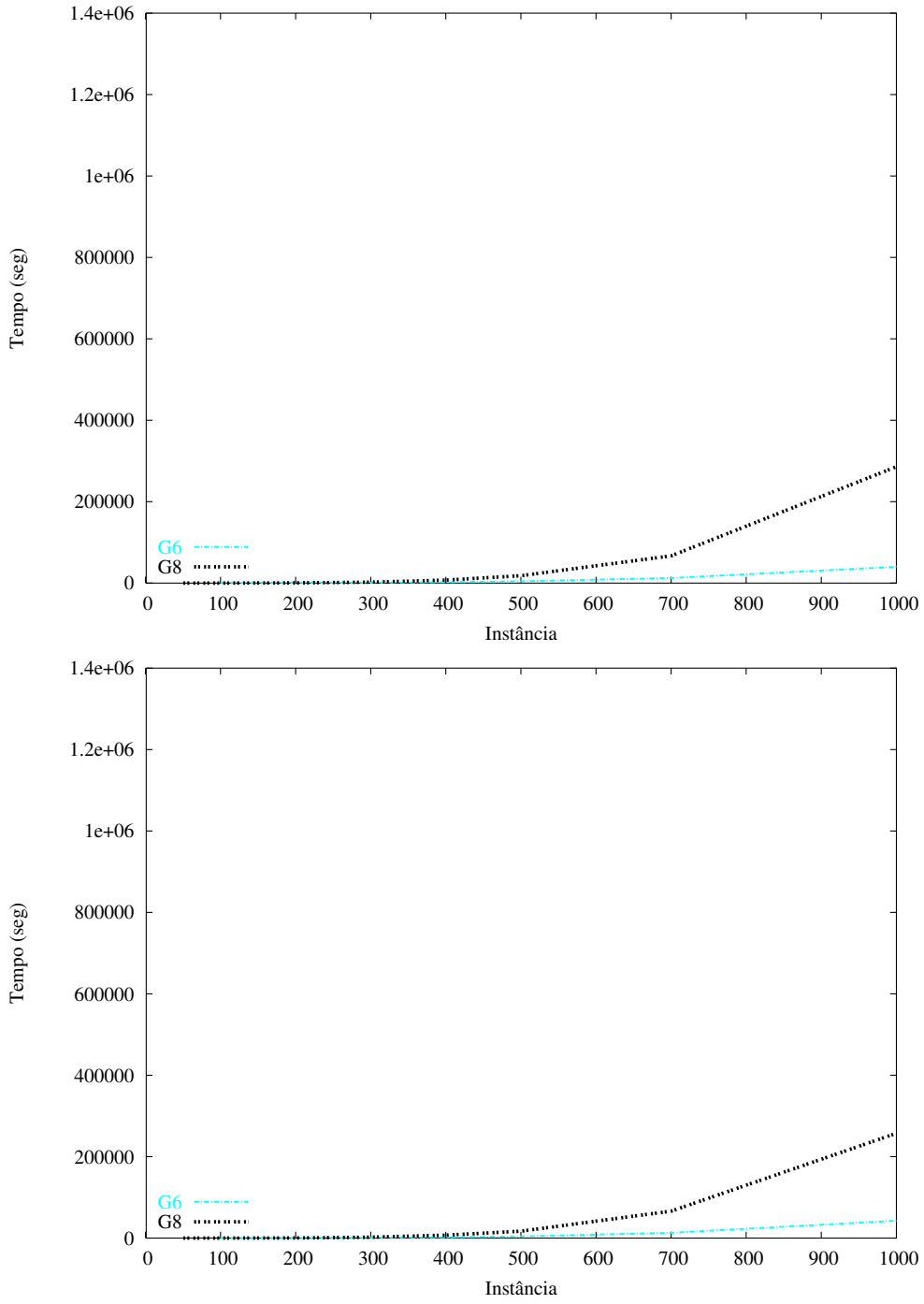

Figura 5.80: Tempo Computacional - versões G6 e G8 para as instâncias do Conjunto de testes A e B, respectivamente

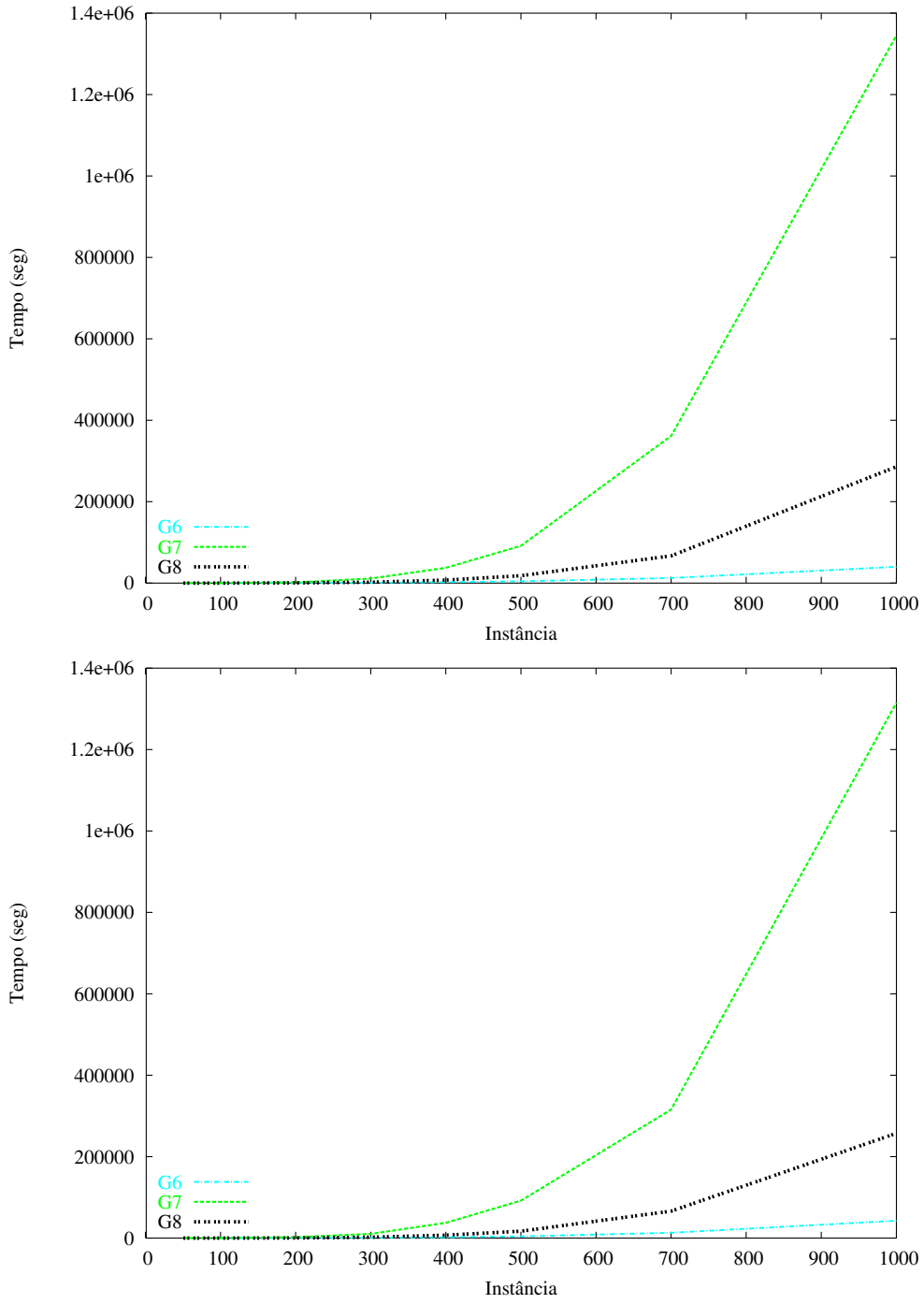

Figura 5.81: Tempo Computacional - versões G6, G7 e G8 para as instâncias do Conjunto de testes A e B, respectivamente

Capítulo 6

Conclusões

Este trabalho apresentou um estudo sobre uma variante do Problema de roteamento de Veículos (PRV), que chamamos de Problema de Roteamento de Sondas de Manutenção (PRSM). Este problema consiste em gerar percursos otimizados para escalonar um conjunto de sondas de manutenção de poços petrolíferos terrestres da região nordeste do Brasil.

Na fase inicial foi feita uma pesquisa bibliográfica sobre o PRSM e foi verificado que existem poucas propostas para solucionar este problema. Assim, outra pesquisa bibliográfica foi realizada, desta vez para analisar problemas similares encontrados na literatura.

Em termos de literatura de Problemas de roteamento e *scheduling* de veículos, o PRSM, embora , apresente algumas similaridades com alguns modelos da literatura ; tal como, o Problema de Roteamento de Veículos com Múltiplos Depósitos (PRMD), na prática existem diferenças significativas entre o PRSM e os modelos existentes. Esta constatação nos fez desistir de simplesmente adaptar modelos já existentes na literatura para se adequar ao PRSM.

O fato de o PRSM ter uma grande importância econômica e, ainda assim, não ser muito explorado na literatura, foi uma motivação para tê-lo como caso de estudo. Porém, a falta de instâncias deste em bibliotecas públicas não nos permitiu fazer comparações com as poucas propostas já existentes.

Sugestões de várias versões para solucionar o PRSM foram então propostas usando como base a estrutura da metaheurística GRASP, que em problemas similares tem alcan-

çado ótimas soluções aproximadas.

Embora o GRASP seja atualmente considerado como uma das melhores metaheuristicas para problemas de otimização combinatória de elevada complexidade computacional (Problemas NP-Completo e NP-Difícil), ao menos nos modelos iniciais, esta técnica possui as iterações totalmente independentes, ou seja, ele não faz uso de nenhum tipo de memória para armazenar informações relevantes obtidas em iterações passadas. Isso faz com que GRASP também seja conhecido como um método *multi-start*.

Por outro lado, outras metaheuristicas eficientes em otimização combinatória, tais como Busca Tabu e Algoritmos Evolutivos, sempre fazem uso de algum tipo de memória para aproveitar informações obtidas em iterações anteriores e assim melhorar o trabalho de busca nas iterações seguintes.

Desta forma, colocamos como proposta deste trabalho desenvolver versões híbridas da heuristisca GRASP, incorporando nestas versões fases de aprendizado e/ou memória, para que informações relevantes de iterações passadas sejam aproveitadas no processo de construção de novas soluções nas iterações remanescentes.

Neste contexto, incluímos dentre os procedimentos propostos: um mecanismo de filtro na etapa de construção do GRASP, Uma Busca Intensiva por Reconexão por Caminhos (*Path Relinking*) e versões com fases de aprendizado para tornar o GRASP adaptativo através do uso de algum tipo de memória.

O objetivo , com isso , foi analisar a influência de cada módulo proposto, desde métodos de construção e busca local, como também módulos adicionais de reconexão por caminhos, filtro e outras estratégias de aprendizado no desempenho final da metaheurística GRASP.

Para efetuar esta análise, iniciamos os experimentos com a comparação das soluções aproximadas obtidas pelo GRASP com as soluções exatas em instâncias de pequenas dimensões do PRSM. Para isso, foi proposta uma formulação matemática do PRSM descrevendo-o como um Problema de Programação Linear Inteira (PPLI), esta utiliza variáveis de fluxo para eliminar a formação de soluções desconexas da origem. Estes testes iniciais foram utilizados para obter soluções exatas para instâncias de pequeno porte, de modo a possibilitar uma avaliação na qualidade das soluções heurísticas ao menos neste

tipo de instâncias.

Como resultado desta análise, verificou-se que o valor ótimo foi encontrado em 88% dos casos e no caso em que o valor ótimo não foi atingido, o valor aproximado encontrado ficou distante apenas 6% do valor ótimo. Entretanto, os tempos computacionais exigidos pelas heurísticas foram extremamente menores do que os exigidos pela solução exata.

Isso projeta o seguinte panorama já conhecido pela literatura: a incapacidade ou limitação de técnicas exatas para a solução de instâncias de grande porte e a viabilidade no uso de heurísticas para estes casos.

Nas demais baterias de testes, as instâncias foram de dimensões maiores, em que o objetivo foi comparar entre elas o desempenho das versões GRASP propostas nesse trabalho. Inicialmente, foram avaliadas as seis versões dos algoritmos GRASP, denominadas de GRASP puro, em que cada uma foi composta por um dentre os dois algoritmos de construção e um dentre os três de busca local propostos neste trabalho.

Na avaliação dos GRASP puros (G1 a G6), observou-se para os conjuntos de testes A (tempo de percurso dominando o fator vazão do poço) e B (vazão dominando o fator tempo de percurso) que buscas locais mais elaboradas fizeram a diferença em relação a qualidade das soluções encontradas. Nestes testes, os algoritmos que utilizam a BL3, baseada em reposicionar os poços da melhor maneira numa rota, obtiveram os melhores desempenhos seguidos pelos algoritmos que utilizam a BL2. Em termos dos algoritmos construtivos, o algoritmo que mais se destacou foi o C2 (poços com maior vazão tendo prioridade de atendimento), ou seja, dentre as versões que utilizam o mesmo algoritmo de busca local, as que utilizam o construtivo C2 foram as que obtiveram os melhores desempenhos nos testes empíricos efetuados.

Em todos os casos, o algoritmo de busca local conseguiu melhorar a solução obtida pelo algoritmo construtivo, mas , em compensação, nestes testes iniciais em que se usa como critério de parada um número máximo de iterações GRASP, os tempos computacionais das versões foram determinados pelos tempos exigidos pelas buscas locais, ou seja, versões com BL3 exigiram, como já era de se esperar, os maiores tempos, seguidos pelas versões que utilizam a BL2, que por sua vez têm tempos maiores do que as versões com BL1.

Posteriormente, foi verificado o impacto da introdução do módulo de Reconexão por

Caminhos (RC) em algumas das versões puras, escolhidas conforme explicações expostas ao longo deste trabalho, criando as primeiras versões híbridas (GRASP+RC). Estas versões com RC se mostraram muito eficazes, encontrando, na maior parte dos testes, resultados de melhor qualidade do que sua versão pura (sem RC). Estes resultados obtidos mostram, mesmo que de forma empírica, que a RC é, sem dúvida, uma das melhores formas de busca intensiva entre duas soluções de boa qualidade. A justificativa para a eficiência da RC pode ser explicada pelo fato deste procedimento usar informações sobre as melhores soluções geradas até o momento pelo algoritmo (conjunto elite). Desta forma, o uso da RC já torna o GRASP adaptativo ou um método com memória.

Entretanto, como desta vez o critério de parada utilizado ainda foi o número máximo de iterações GRASP, os tempos computacionais das versões com RC, como já era de se esperar, se mostraram os maiores, já que o trabalho por iteração das versões com RC é maior do que o das versões puras.

Posteriormente, apresentamos outras formas de gerar versões adaptativas do GRASP, em que, basicamente, nas primeiras iterações é feito um treinamento com diferentes métodos de construção, busca local e calibração de outros parâmetros e após esta fase de treinamento, as melhores combinações de métodos e/ou parâmetros são fixados para as iterações restantes do GRASP. A meta que colocamos ao propor estas versões adaptativas é construir versões robustas do GRASP, ou seja, uma metaheuristica que não apresente soluções muito heterogêneas nas suas várias execuções de uma dada instância. Esta regularidade nestas heurísticas as tornam mais confiáveis e com isso mais aptas a serem usadas na prática.

Finalmente, em uma etapa seguinte, analisamos o impacto da inserção de um módulo de filtro em algumas das melhores versões analisadas até o momento. Os resultados mostraram que o filtro tende a melhorar o desempenho final do GRASP associado em termos da qualidade da solução gerada sem onerar significativamente os tempos computacionais exigidos.

Em uma outra etapa, efetuamos um estudo de análise da eficiência de cada versão aqui proposta sob um outro critério de parada, critério este que não prejudique nem as versões com iteração muito pesada (como GRASP com busca BL3, ou versões com RC) na análise do tempo computacional exigido e nem as versões muito leves no tocante à

qualidade das soluções geradas; algo que pode ocorrer usando como critério de parada um numero máximo de iterações GRASP.

Para isso, usamos como critério de parada o alcance de um valor alvo pelas heurísticas. Além disso, nesta nova bateria de testes, colocamos como meta analisar a robustez de cada versão GRASP aqui proposta através de um número elevado de execuções de cada uma das instâncias analisadas. Este processo, que chamamos de análise probabilística empírica, foi então adotado.

O resultado da análise probabilística empírica mostrou uma nítida superioridade das versões com busca local mais sofisticada (como aquelas que usam a busca BL3) e daquelas que incorporam módulos de reconexão por caminhos (RC), agora também em termos de velocidade de convergência, chegando quase sempre no valor alvo estipulado com tempos computacionais similares ou até menores ao das versões puras com iterações mais leves. Comparando o desempenho das versões puras com aquelas incluindo reconexão por caminhos (GRASP+RC), verificamos que as versões GRASP+RC são, na maioria das vezes, mais rápidas que versões puras no tocante à convergência para valores alvos pré-definidos. Essa diferença se acentua, ainda mais, quando trabalhamos com alvos mais difíceis de serem alcançados (alvos de melhor qualidade). Além disso, os testes mostraram que versões mais sofisticadas, como as com PR, quase sempre atingem o valor alvo, ao contrário de alguns versões puras que muitas vezes não conseguem. Neste contexto, já podemos vislumbrar uma robustez maior das versões híbridas mais sofisticadas, como por exemplo aquelas que incluem busca local BL3, RC e filtro.

Em relação às versões adaptativas com fase de treinamento, estas se mostraram muito eficazes em relação às demais versões no tocante a sua regularidade. Ou seja, estas sempre produzem soluções de boa qualidade, ao contrário de algumas versões puras que ora encontram soluções muito boas, ora muito fracas.

Desta forma, os resultados efetuados mostram que o desempenho da heurística GRASP é influenciado pela qualidade da solução na fase de construção (por exemplo, concluímos que é interessante usar o módulo do filtro) e por uma busca local eficiente (tipo a BL3, que resulta em melhoras significativas). Mostram também que módulos adicionais de busca intensiva, tipo RC, são fortemente recomendados para qualquer versão GRASP. Outro aspecto observado é que as versões híbridas incluindo um conjunto de procedimen-

tos ou somente um destes, tais como filtro, busca local tipo BL3 e RC são muito estáveis, mostrando uma robustez visivelmente melhor que as demais versões puras.

Finalmente, as versões adaptativas com fase de treinamento mostraram ser as mais robustas e sempre gerar soluções de alta qualidade quando comparada com as soluções das demais versões. Acreditamos que, com isso, atingimos as metas estabelecidas neste trabalho, ou seja, desenvolver formas eficientes e robustez da metaheuristica GRASP para o PRSM.

Como trabalhos futuros, acreditamos que pesquisas nesta linha de tornar o GRASP cada vez mais adaptativo, tanto para versões sequênciais como paralelas, possam produzir bons resultados para o problema aqui analisado e para os demais problemas da otimização combinatória. Em relação ao PRSM, acreditamos que soluções obtidas com uma metaheurísticas Busca Tabu eficiente possa, também, produzir bons resultados aproximados já que esta é a técnica heurística que tem apresentado os melhores resultados para os modelos de problemas de roteamento de veículos da literatura.

Referências

[1] DALBONI, F. L. *Algoritmos Evolutivos Eficientes para um Problema de Roteamento de Veículos*. Dissertação (Mestrado) — Universidade Federal Fluminense, 2003.

[2] LIMA, F. C. de. *Otimização das Intervenções em Poços de Petróleo por Sondas de Produção Terrestre: Uma abordagem Metaheurística*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2002.

[3] ALOISE, D. J. et al. Scheduling workover rigs for onshore oil production. *Discrete Applied Mathematics*, 2004.

[4] ALOISE, D. et al. Heurística de colônia de formigas com path-relinking para o problema de otimização da alocação de sondas de produção terrestre. *Em Anais do XXXIV Simpósio Brasileiro de Pesquisa Operacional (SBPO), Rio de Janeiro*, 2002.

[5] RENAUD, J.; LAPORTE, G.; BOCTOR, F. F. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers Operational Research Vol. 23 Number 3*, p. 229–235, 1996.

[6] TILLMAN, F. A. The multiple terminal delivery problem with probabilistic demands. *Transp. Sci. 3*, p. 192–204, 1969.

[7] WREN, A.; HOLLIDAY, A. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Opns Res. Q. 23*, p. 333–344, 1972.

[8] DESROCHERS, M.; LENSTRA, J. K.; SAVELSBERGH, M. W. P. Theory and methodology: A classification scheme for vehicle routingn and scheduling problems. *European Journal of Operational Research 46*, p. 322–332, 1990.

[9] SALHI, S.; NAGY, G. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society 50*, p. 1034–1042, 1999.

[10] MIN, H.; CURRENT, J.; SCHILLING, D. The multiple depot vehicle routing problem with backauling. *Journal of Business Logistics, ABI/INFORM Global*, p. 259–288, 1992.

[11] LAPORTE, G. et al. Classical and modern heuristics for the vehicle routing problem. *Intl. Trans. in Op. Res. 7*, p. 285–300, 2000.

[12] RIBEIRO, C. C.; SOUMIS, F. A column generation approach the multiple-depot cehicle scheduling problem. *Operationals Research 42*, p. 41–52, 1994.

[13] COSTA, L. R. da. *Soluções para o Problema de Otimização de Itinerários de Sondas*. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2005.

[14] BALL, M. O. et al. Network routing. *In Handbooks in Operations Research and Management Science vol 8*, 1995.

[15] HANSEN, P.; MLADENOVI'C, N. An introduction to variable neighbourhood search. *Metaheuristics: Advances and trends in local search procedures for optimization*, p. 433–458, 1999.

[16] HANSEN, P.; MLADENOVI'C, N. A tutorial on variable neighborhood search. *Technical report, Les Cahiers du GERAD G-2003-46, Montr'eal*, 2003.

[17] HANSEN, P.; MLADENOVI'C, N. Variable neighborhood search. *Handbook of Metaheuristics - Kluwer Academic Publishers*, p. 145–184, 2003.

[18] HANSEN, P.; MLADENOVI'C, N. Variable neighbourhood search. *Computers and Operations Research 24*, p. 1097–1100, 1997.

[19] GLOVER, F. Heuristic for integer programming using surrogate constraints. *Decision Sci. 8*, p. 156–166, 1977.

[20] CHRISTOFIDES, N.; EILON, S. An algorithm for one vehicle-dispatching problem. *Opl Res. Q. 20*, p. 309–318, 1969.

[21] GILLETT, B. E.; JOHNSON, J. G. Multi-terminal vehicle-dispatch algorithm. *Omega 4*, p. 711–718.

[22] CHAO, I. M.; GOLDEN, B. L.; WASIL, E. A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions. *Am. J. Math. Mgmt Sci. 13*, p. 371–406, 1993.

[23] CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Opns Res. 46*, p. 93–100, 1964.

[24] CASCO, D. O.; GOLDEN, B. L.; WASIL, E. A. Vehicle routine with backhauls: models, algorithms and case studies. *Vehicle Routing: Methods and Studies*, p. 127–147, 1988.

[25] FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *J. of Global Optimization 6*, p. 109–133, 1995.

[26] ROSSETI, I. C. M. *Estratégias seqüenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio De Janeiro, 2003.

[27] RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. *Handbook of Metaheuristics*, p. 219–249, 2003.

[28] LAGUNA, M.; JR, F. L.; ALOISE, D. Um algoritmo heurístico guloso aplicado ao problema do gerenciamento das intervenções em poços petrolíferos por sondas de produção terrestre. *Em Anais do XXXIII Simpósio Brasileiro de Pesquisa Operacional (SBPO), Campos do Jordão*, p. 44–52, 2001.

[29] PRAIS, M.; RIBEIRO, C. C. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing 12*, p. 164–176, 2000.

[30] FLEURENT, C.; GLOVER, F. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing 11*, p. 198–204, 1999.

[31] SILVA, G. C.; OCHI, L. S.; MARTINS, S. L. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. *Third International Workshop on Experimental and Efficient Algorithms*, p. 498–513, 2004.

[32] FESTA, P.; RESENDE, M. G. C. Grasp: An annotated bibliography. *Essays and surveys in metaheuristics, C.C. Ribeiro and P. Hansen, Eds., Kluwer Academic Publishers*, p. 325–367, 2002.

[33] CHRISTOFOLETTI, L. M. *Métodos de Reinício Aplicados ao Sequenciamento em Uma Máquina com Tempos de Preparação e Datas de Entrega*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2002.

[34] BERTOSSI, A. A.; CARRARESI, P.; GALLO, G. On some matching problems arising in vehicle scheduling models. *Networks 17*, p. 271–281, 1987.

[35] AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. Probability distribution of solution time in grasp: an experimental investigation. *Journal of Heuristics 8*, p. 343–373, 2002.