Uma nova modelagem para o problema de escalonamento de tarefas com restrições de recursos

André Renato Villela da Silva

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

Orientador: Luiz Satoru Ochi

Niterói, Fevereiro de 2006.

Uma nova modelagem para o problema de escalonamento de tarefas com restrições de recursos

André Renato Villela da Silva

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

Aprovada por:

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

Profa. Maria Cristina Silva Boeres / IC-UFF

Profa. Maria Claudia Silva Boeres / INF-UFES

Niterói, Fevereiro de 2006.

A minha querida família e meus amigos pelo apoio e pela paciência de me aturarem.... e aos governantes que procuram investir seriamente em educação neste país (são bem poucos, é claro).

Agradecimentos

A Deus, pelos dons da vida, do amor, da saúde, por retirar os *bugs* do compilador.... e por todas as outras coisas.

A minha família, pelo incentivo, pelo carinho e pela comidinha gostosa...

Aos meus amigos, minhas fontes de alegria e esperança, que me fizeram decorar a definição destes problemas (de tanto que eu tive de repetir para eles).... obrigado pelos sorrisos amarelos de quem entenderam tudo....

A todos os professores que se dedicam verdadeiramente a este sublime ofício do magistério.

A FAPERJ - Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro - pela bolsa de mestrado que me foi concedida para a realização deste curso, sob o número E-26/150.575/2004.

A todas as demais pessoas que passaram pela minha vida, meu muito obrigado por tudo o que deixaram de si e pelos inesquecíveis momentos que vivemos.... estaremos todos juntos um dia....

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

Uma nova modelagem para o problema de escalonamento de tarefas com restrições de recursos

André Renato Villela da Silva

Fevereiro/2006

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

Este trabalho apresenta uma nova modelagem a ser utilizada no problema de escalonamento de tarefas com restrições de recursos (PETRR). Alguns modelos de PETRR adotam um sistema de recursos renováveis dentro de um horizonte de planejamento composto de um conjunto de períodos, isto é, no decorrer do problema, após a ativação de tarefas, estas passam a gerar receitas em quantidades fixas ou variáveis a cada período subseqüente. No modelo aqui proposto, a partir do momento em que se ativa uma tarefa até o último período considerado, uma quantidade de recursos denominada lucro (associada à tarefa ativada) é disponibilizada a cada período. Assim, a quantidade de recursos disponíveis, num dado período, vai depender de quais tarefas foram ativadas até então e de quando isto ocorreu. Esta modelagem reflete, de maneira mais real, grandes projetos de expansão de empresas, que podem ser feitos em etapas e que já admitem retornar algum lucro antes do término do projeto como um todo. Foram propostas algumas heurísticas para tratar este problema incluindo conceitos das meta-heurísticas GRASP e Algoritmos Evolutivos (AEs). O trabalho mostra como determinados parâmetros do problema podem ser calibrados bem como formas de combinar de modo eficiente heurísticas de construção e busca local numa estrutura GRASP e/ou AE.

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

A new modelling for the resource-constrained task scheduling problem

André Renato Villela da Silva

February/2006

Advisors: Luiz Satoru Ochi
Department: Computer Science

This work presents a new modelling to be used on resource-constrained task (project) scheduling problems (RCTSP or RCPSP). Some RCTSP(RCTPSP) models use a renewable resource approach within a planning horizon composed of periods, i.e. in the course of the problem, after the tasks activation, these can generate budget in fixed or variable amount at each following period. On the proposed modelling, from the moment when a task is activated until the last considered period, a quantity of resources called profit (associated to the activated task) is available at each period. Thus, the quantity of available resources, at a given period, will depend on what tasks were activated until this period and when it happened. This modelling reflects, in a more realistic manner, big company expansion projects, that can be made in steps and allow, at once, to get some profits before the project conclusion. Some heuristics were proposed to handle this problem including concepts of GRASP and Evolutive Algorithms (EAs) meta-heuristics. This work shows how certain problem parameters can be calibrated and ways to combine, efficiently, construction heuristics and local searches, in a GRASP and/or EA structure.

Palavras-chave

1. Problema de Escalonamento de Tarefas com Restrições de Recursos
2. Meta-heurística GRASP
3. Meta-heurística Algoritmo Evolutivo
4. Heurísticas de construção e busca local

Glossário

GRASP	:	*Greedy Randomized Adaptive Search Procedure*;
AG	:	Algoritmo Genético;
AE	:	Algoritmo Evolutivo;
PET	:	Problema de Escalonamento de Tarefas;
PETRR	:	Prob. de Escal. de Tarefas com Restrições de Recursos;
PETRRD	:	Prob. de Escal. de Tarefas com Restrições de Recursos Dinâmicos;
LRC	:	Lista Restrita de Candidatos;
MKP	:	*Multiple Knapsack Problem*;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O problema de escalonamento de tarefas (PET) é muito conhecido e muito estudado já há bastante tempo. Consiste basicamente num conjunto de tarefas que devem ser executadas (ativadas, iniciadas, ou outro termo semelhante) de alguma forma, no menor tempo possível. O modelo clássico pressupõe, além das tarefas, precedências entre elas e unidades de processamento (UP) nas quais devem ser executadas as tarefas. Estas unidades podem ter características semelhantes (ambiente homogêneo) ou distintas (heterogêneo) [1]. Deve-se escolher, portanto, em qual UP cada tarefa será executada e em que momento isto ocorrerá. Alguns modelos levam em conta o tempo gasto na comunicação entre tarefas que forem alocadas a UPs distintas [2].

Uma outra modelagem também muito estudada é chamada de máquina de estados. Nela, há uma única unidade de processamento que pode transitar por entre um conjunto finito de estados. Pode-se dizer que cada estado altera, viabiliza ou inviabiliza a execução das tarefas de alguma forma. Este modelo é conhecido como multi-modo ou multi-modal, por existirem vários estados nos quais as tarefas poderão ser executadas [3].

Geralmente, o objetivo do problema é executar todas as tarefas no menor tempo possível (*makespan*), inclusive com mais de um conjunto de tarefas (chamado, às vezes, de projeto) sendo tratado simultaneamente [4, 5]. Mas outros objetivos também podem ser considerados como: evitar que a unidade de processamento fique muito tempo inoperante, ter a maior vazão possível (executar o maior número de tarefas num determinado prazo), atender da melhor forma possível tarefas com prioridades diferentes, entre várias outras.

Generalizando um pouco mais o problema original de escalonamento de tarefas, pode-se pensar que para a execução das mesmas é necessário pagar um custo (quantidade de

recursos). O recurso gasto na execução da tarefa deve ser retirado de um montante disponibilizado para tanto. Também é comum encontrar casos onde esses recursos são de tipos diferentes [6, 7]. Ou seja, existe mais de um tipo de recurso necessário para a execução de uma tarefa; por exemplo: a construção de uma chave necessita exclusivamente de uma pequena quantidade de metal, já uma casa precisa de tijolos, madeira, cimento em diversas quantidades. De qualquer forma, para que uma tarefa seja executada, é imprescindível que estes recursos estejam todos disponíveis no momento da execução da tarefa em questão. De maneira geral, estes recursos podem ser classificados como renováveis (quando os recursos gastos na ativação de uma tarefa podem ser repostos) e não-renováveis (quando não puderem ser repostos) [6].

Estas modelagens podem ser consideradas estáticas, pois não admitem alteração nas características e nos tempos em que as tarefas podem ser ativadas, já que estes são definidos antes do início do problema (dados de entrada) e valem até o final do mesmo. No entanto, existem também modelagens chamadas de dinâmicas, pois admitem estas alterações. Estas tentam refletir melhor certos problemas onde novas tarefas podem ser adicionadas (ou ter suas características e/ou tempos de início alterados) e devem ser escalonadas da mesma forma que as já existentes [8]. Geralmente, nestes casos, procura-se alterar o menos possível o plano de escalonamento original, para se evitar a repetição dos cálculos já realizados.

Aplicações práticas do problema de escalonamento de tarefas são muito facilmente encontradas em várias sub-áreas da Otimização Combinatória, como podem ser vistas em [6, 7, 9, 10, 11, 12].

Neste trabalho é apresentada uma nova modelagem para o problema de escalonamento de tarefas, chamado PETRRD (Problema de Escalonamento de Tarefas com Restrição de Recursos Dinâmicos). Para sua solução, são propostas, além do modelo matemático para obter soluções exatas, algumas versões das meta-heurísticas GRASP e Algoritmos Evolutivos com o objetivo de gerar bons limites inferiores de um valor ótimo. Estas meta-heurísticas foram escolhidas por apresentarem filosofias distintas (a primeira é do tipo *restart* e a segunda é da classe das heurísticas de população) e serem bastante difundidas no meio científico. Outras técnicas poderão ser vistas em [9, 13, 14], por exemplo.

Também são apresentados mecanismos chamados de Técnicas Auxiliares que se pro-

põem a colaborar com as heurísticas construtivas para a obtenção de resultados melhores. Estas técnicas se baseiam em características tanto da modelagem proposta como dos grafos de entrada e são, de maneira geral, procedimentos muito simples (computacionalmente).

O restante deste trabalho está organizado da seguinte forma: no Capítulo 2, são explicadas as características do modelo proposto; no Capítulo 3, são mostradas algumas técnicas que serão utilizadas nos algoritmos e as heurísticas propostas; as instâncias que foram geradas para este problema, são mostradas no Capítulo 4; no Capítulo 5, são apresentados os resultados computacionais obtidos e uma análise mais detalhada dos mesmos; no Capítulo 6, são apresentadas algumas conclusões obtidas a partir dos experimentos realizados e, por fim, são discutidos possíveis trabalhos futuros referentes a este tema.

Capítulo 2

A Modelagem Proposta - PETRRD

A modelagem aqui proposta difere de outras mais comumente encontradas na literatura, principalmente, no que se refere à forma como os recursos se renovam ao longo do problema. Por isso, resolveu-se chamá-la de Problema de Escalonamento de Tarefas com Restrições de Recursos Dinâmicos (PETRRD). Antes, porém, de descrever tal modelagem, cabe explicar o que motivou a propô-la.

2.1 Motivação

Jogos de computador são, entre os diversos tipos de *software*, os que mais se utilizam de recursos de inteligência artificial, na grande maioria das vezes. Entre os diversos tipos de jogos existentes, aqueles classificados como de estratégia ("de tempo real" ou "em turnos") freqüentemente envolvem problemas como alocação de recursos, recobrimento de rotas, escalonamento de tarefas, e vários outros problemas comuns à área de Otimização Combinatória.

Um destes jogos, chamado *Civilization III*®, da empresa *FIRAXIS Games, INC.*, mereceu especial atenção. Neste jogo, o usuário (jogador) assume o papel de líder de uma nação muito pouco desenvolvida e deve fazê-la progredir tentando alcançar determinado estágio, onde será decretada a vitória para esta civilização. Além da civilização controlada pelo jogador, existem ainda, simultaneamente, outras civilizações controladas pelo computador, ou seja, por programas de computador que se utilizam de técnicas de inteligência artificial para determinar as ações a serem tomadas. Neste jogo, ainda é possível interagir com as demais civilizações por meios diplomáticos, comerciais ou até mesmo militares.

Cada civilização é composta de cidades (que devem possuir várias benfeitorias como: hospitais, escolas, usinas geradoras de energia, quartéis) e unidades móveis (civis e militares). Além disso é necessário planejar a captação de recursos (minerais, metais, entre outros), efetuar uma expansão cultural e territorial da sua civilização e fazer alianças e táticas militares de ataque e defesa. Sendo assim, é possível se pensar em dezenas de estratégias que poderiam ser utilizadas para se alcançar um determinado objetivo desejado.

Entretanto, o aspecto mais difícil de planejar é, talvez, o progresso tecnológico de sua nação. O jogador deve especificar qual "tecnologia" ele pretende que sua civilização descubra, como por exemplo: a roda, a escrita, a confecção em cerâmica, a matemática e a física. Para que cada descoberta seja feita é necessário, basicamente, que a civilização fique alguns turnos pesquisando tal tecnologia, sendo que apenas uma tecnologia pode ser pesquisada por vez. Quando a tecnologia é descoberta, ela permanece favorecendo sua civilização geralmente até o fim do jogo (algumas tecnologias fazem com que outras tecnologias se tornem obsoletas).

Este benefício trazido pela descoberta de uma tecnologia pode possibilitar a construção de novos equipamentos para as cidades, a construção de novas unidades civis e/ou militares, entre outras coisas. Além de possibilitar que novas tecnologias sejam descobertas, criando assim uma espécie de precedência entre tecnologias. A Figura 2.1 mostra uma tela do jogo, com algumas indicações.

Independentemente da estratégia adotada para vencer o jogo, o jogador sempre irá se deparar com este "problema". E uma boa escolha das tecnologias a serem descobertas será fundamental para que sua estratégia seja posta em prática, uma vez que no início do jogo, as possibilidades de vitória são muito pequenas. Para um jogador que planeje atuar de forma militar, a escolha de tecnologia que envolvam armamentos parece melhor. Para um jogador que precise expandir rapidamente seu território, investir em tecnologias que garantam alimentos e aumento populacional seria a atitude mais correta. Enfim, o benefício trazido pela descoberta de uma tecnologia é bastante subjetivo, sendo mais ou menos importante para uma estratégia ou para outra.

Desta forma, é proposta uma modelagem para um problema de escalonamento de tarefas que tivesse semelhança com este problema de "descobertas de tecnologias": as tecnologias serão chamadas de nós ou tarefas; o esforço gasto na descoberta destas tecnologias

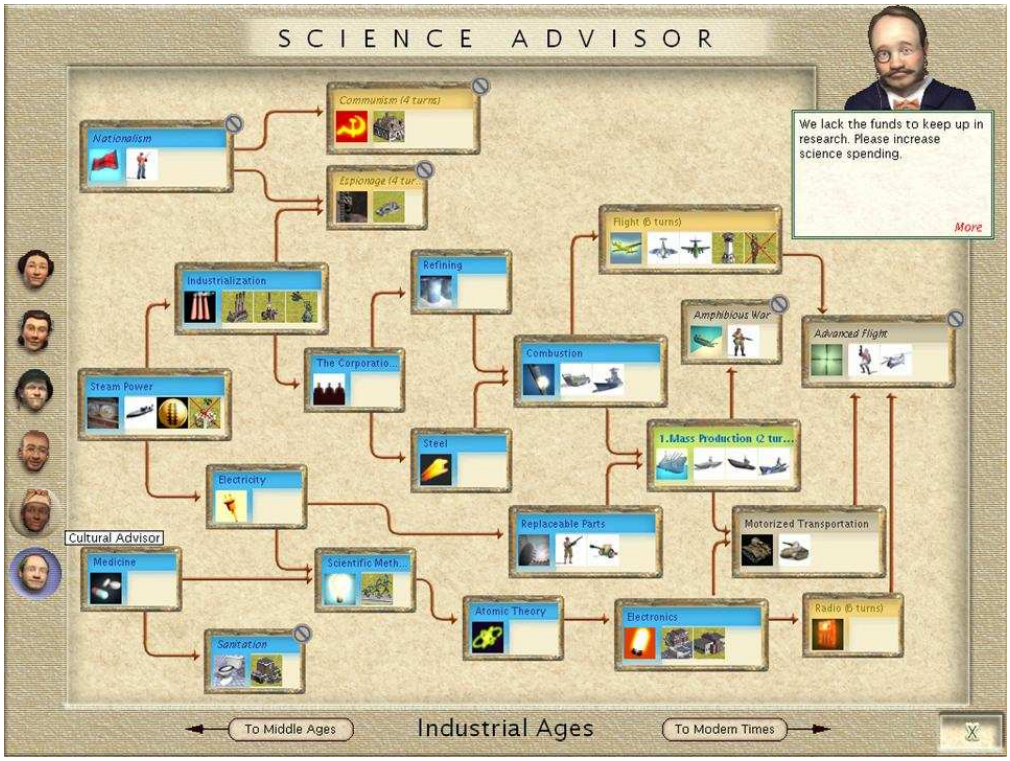


Figura 2.1: Tela do jogo *Civilization III*®

será chamado de recurso(s) (a idéia de gastar tempo para a descoberta da tecnologia será reformulada para se tornar um gasto de recurso(s)); o benefício trazido pela tecnologia será chamado de lucro (também será necessário quantificar o benefício, ao invés de tratá-lo de forma subjetiva). Uma modelagem mais detalhada é ser vista a seguir.

2.2 O modelo proposto

Suponha um grafo direcionado $G = (V, A)$, com $|V| = n$. A cada vértice i (uma tarefa a ser escalonada) de V está associado um custo positivo c_i e um lucro l_i. Para cada vértice j, os vértices i tais que o arco $(i, j) \in A$ formam o conjunto dos predecessores de j, que será representado por $P(j)$.

Suponha também que o intervalo de tempo no qual as tarefas devem ser escalonadas está discretizado num total de T unidades de tempo (dias, semanas, meses). O objetivo é escolher quais tarefas escalonar e quando fazê-lo para que ao final das T unidades de tempo (em $T + 1$), haja a maior quantidade de recursos disponíveis.

Embora no problema sejam supostas T unidades de tempo, será considerado um

instante de tempo $T+1$ no qual não poderão ser ativadas as tarefas. Este instante $T+1$ servirá apenas para efeito de cálculo do resultado do problema, pois o lucro obtido em T só será retornado no instante de tempo seguinte. Desta forma, a quantidade de recursos que ficarem disponíveis no inicío deste tempo $T+1$ será a solução do problema.

Para cada tarefa i escalonada é preciso pagar o custo c_i associado a ela (retirando-se o mesmo do montante de recursos disponíveis no momento). A partir da ativação desta tarefa i, os recursos disponíveis serão acrescidos de l_i unidades a cada unidade de tempo subseqüente (mais precisamente ao final de cada instante de tempo, desde o tempo em que foi ativado até o final do horizonte de tempo). Este incremento de recursos é o principal diferenciador deste modelo proposto em relação aos existentes na literatura. Em [3], por exemplo, para cada instante de tempo do problema, existe uma quantidade de recursos fixa e invariável.

Ao propor que esta quantidade de recursos seja decorrente de cada tarefa ativada e que este lucro seja incorporado ao problema a cada instante de tempo posterior à ativação da tarefa, está sendo gerada uma modelagem que pode ser entendida como um grande projeto de uma empresa dividido em sub-projetos (tarefas), que têm relação de precedência entre si. Cada sub-projeto tem um custo de implementação (c_i, da tarefa i) e que a partir de então retorna um lucro l_i à empresa, a cada instante de tempo subseqüente a sua ativação.

Para deixar o modelo proposto mais claro, é necessária a explicação de mais alguns conceitos, que ajudarão no entendimento do problema. Estes conceitos são descritos a seguir.

2.3 Alguns conceitos

Ativação: é a incorporação de uma tarefa i na solução do problema. Para tanto, paga-se um custo c_i e obtém-se, a cada instante de tempo subseqüente, um lucro l_i.

Tarefa disponível: é a tarefa i que não possui predecessores ($|P(i)| = 0$) ou cujos predecessores já tenham sido ativados. A tarefa i só estará disponível se houver recursos suficientes para ativá-la.

Horizonte de tempo: é o conjunto de unidades de tempo (discretizado) no qual as

tarefas poderão ser ativadas. Varia de 1 (um) até T (dado do problema).

Período de ativação: é a quantidade de unidades de tempo, desde a ativação da tarefa até o fim do horizonte de tempo (T).

Custo (simples) c_i: é a quantidade de recursos necessária para a ativação de uma tarefa i.

Custo composto cc_i: é a quantidade de recursos necessária para a ativação de uma tarefa i e de todos os seus predecessores.

Lucro (simples) l_i: é quantidade de recursos que são gerados por uma tarefa i, em cada instante do período de ativação.

Custo-benefício: é a divisão do custo da tarefa i pelo seu respectivo lucro ($\frac{c_i}{l_i}$).

Lucro total: é o produto do lucro pela quantidade de unidades de tempo do período de ativação.

Lucro líquido: é a diferença entre o lucro total e o custo de uma tarefa. Se o lucro líquido for positivo (>0), diz-se que é um lucro real.

Recursos disponíveis (em t) Q_t: é a quantidade de recursos que poderão ser utilizados no instante t. Quando uma tarefa i é ativada, decrementa-se c_i de Q_t. Quando se inicia um instante de tempo t, por exemplo, o valor de Q_t é igual ao valor de Q_{t-1} (os recursos que sobraram do instante anterior) somado a L_{t-1} (lucro do instante de tempo anterior).

Lucro de tempo (em t) L_t: é a quantidade de recursos incorporados aos recursos disponíveis, no início do instante t. Ou seja, ao iniciar um instante t, o valor de L_t é igual ao de L_{t-1}. Conforme as tarefas vão sendo ativadas em t, este valor vai sendo incrementado de acordo com o lucro l_i das tarefas i que estão sendo ativadas.

2.4 Formulação matemática do PETRRD

Neste trabalho é proposta uma formulação matemática do PETRRD, descrevendo-o como um Problema de Programação Linear Inteira. A variável binária x_{it} recebe valor 1, se a tarefa i for ativada no tempo t e valor zero caso contrário. A variável Q_t indica a quantidade de recursos disponíveis no início do instante t (Q_1 é dado de entrada do

problema), enquanto a variável L_t representa o lucro de tempo que será incorporado aos recursos no tempo seguinte $(t+1)$, com $L_0 = 0$. A formulação matemática pode ser vista a seguir:

Maximizar:

$$Q_{T+1} \tag{2.1}$$

Sujeito à:

$$|P(i)|x_{i1} = 0 \quad \forall i = 1, ..., n \tag{2.2}$$

$$|P(i)|x_{it} \leq \sum_{j \in P(i)} \sum_{t'=1}^{t-1} x_{jt'} \quad \forall i = 1, ..., n \quad \forall t = 2, ..., T \tag{2.3}$$

$$\sum_{i=1}^{n} c_i x_{it} \leq Q_t \quad \forall t = 1, ..., T \tag{2.4}$$

$$Q_{t+1} = Q_t - \sum_{i=1}^{n} c_i x_{it} + L_t \quad \forall t = 1, ..., T \tag{2.5}$$

$$L_t = L_{t-1} + \sum_{i=1}^{n} l_i x_{it} \quad \forall t = 1, ..., T \tag{2.6}$$

$$\sum_{t=1}^{T} x_{it} \leq 1 \quad \forall i = 1, ..., n \tag{2.7}$$

$$x_{it} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall t = 1, ..., T \tag{2.8}$$

$$Q_t, L_t \in \mathbb{N} \quad \forall t = 1, ..., T + 1 \tag{2.9}$$

Em (2.1) está a função objetivo do problema que é a de maximizar a quantidade de recursos no instante imediatamente posterior ao fim do horizonte de tempo $(T + 1)$. As inequações (2.2) garantem que uma tarefa i só será ativada no primeiro instante de tempo se ela não possuir predecessores ($|P(i)| = 0$). As inequações (2.3) modelam a precedência existente entre uma tarefa i e seus predecessores $P(i)$: isto significa que se uma tarefa i é ativada num tempo $t, t > 1$, então todos os seus predecessores devem ter sido ativados até o intervalo de tempo imediatamente anterior $(t-1)$. Em (2.4) está expressa a restrição de recursos que estão disponíveis para a ativação das tarefas num instante t: o total dos custos das tarefas a serem ativadas em t não pode ultrapassar a quantidade de recursos disponível neste instante. As restrições (2.5) mostram como é feito o incremento de recursos para um instante $(t + 1)$, a partir de t: é a soma dos recursos inicialmente disponíveis em

t, subtraído dos custos para a ativação das tarefas neste intervalo e somado ao lucro acumulado até então (L_t). Vale observar que o valor inicial dos recursos disponíveis Q_1 é um dado do problema. De forma semelhante, as equações (2.6) modelam o incremento do lucro como a soma do lucro acumulado anteriormente (L_{t-1}) e do lucro obtido pelas tarefas ativadas, também no instante anterior. O lucro inicial L_0 é considerado nulo. Em (2.7) são descritas as restrições que impedem que uma mesma tarefa possa ser ativada mais de uma vez. Em (2.8) está garantido o domínio binário das variáveis x_{it} e em (2.9) o domínio natural das variáveis Q_t e L_t.

Para ilustrar alguns destes conceitos será mostrada a figura 2.2, a seguir. Em branco, estão as tarefas ativadas; em cinza, as disponíveis no instante de tempo em questão; em preto as não-disponíveis. Os números acima das tarefas indicam respectivamente o custo e o lucro da tarefa. Seja suposto que $Q_1 = 2$ e $L_1 = 0$, inicialmente. O horizonte de tempo é $T = 3$, ou seja, até a parte (c) da figura.

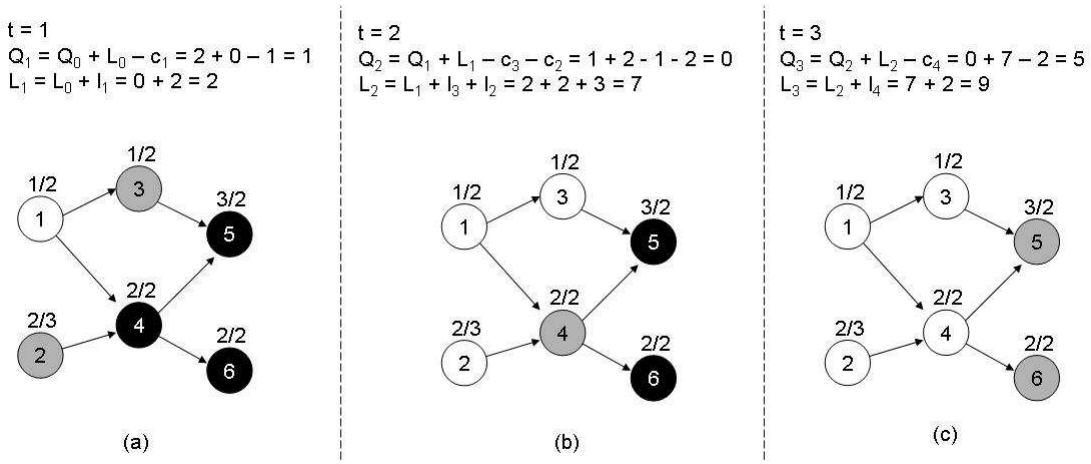


Figura 2.2: Construção de uma solução

Na parte (a), havia duas tarefas disponíveis (1 e 2), por não terem predecessores. Foi feita a escolha da tarefa 1. Esta passou a ser ativada (cor branca). Assim, o custo da tarefa 1 (1 unidade) foi decrementado dos recursos disponíveis Q_1 e o lucro L_1 foi acrescido de 2 unidades (lucro da tarefa 1). A tarefa 3 que só dependia da ativação da tarefa 1 passou a ser disponível (cor cinza).

Em (b), pode ser visto a continuação do processo: o lucro L_2 começa com o mesmo valor do término de $t = 1$ (2 unidades). Os recursos disponíveis, entretanto, são obtidos pelo que veio de $t = 1$ (1 unidade) somado ao lucro L_1 (2 unidades). As tarefas 2 e 3 são

ativadas neste instante; seus custos decrementados de Q_2 e seus lucros somados a L_2. Os estados das tarefas também são atualizados (a tarefa 4 passa a estar disponível; a tarefa 5 continua não disponível porque a tarefa 4 ainda não foi ativada).

Finalmente na parte (c), os valores de Q_3 e L_3 são calculados e a tarefa 4 é escolhida para ser ativada. Todos os valores e estados das tarefas são atualizados. Como $t = T$, encerra-se o processamento. O valor desta solução encontrada correspondente a Q_4 (Q_{T+1}) é facilmente calculado de forma semelhante: soma-se o que vem de Q_3 (5 unidades) com o lucro L_3 (9 unidades) totalizando 14 unidades.

Capítulo 3

Heurísticas para o modelo proposto

Pelo fato do modelo PETRRD estar sendo proposto neste trabalho, não existem, desta forma, outros trabalhos associados a este modelo. Modelos da literatura que mostram alguma similaridade com o proposto são aqueles já mencionados nos capítulos anteriores deste trabalho.

O PETRRD tem como um caso particular a versão onde não existe precedência entre tarefas. Neste caso, o problema resultante pode ser visto como um problema de múltiplas mochilas (*MKP - multiple knapsack problem*), que é classificado como NP-Completo [15]. Desta forma, o PETRRD, no seu caso geral, dificilmente poderá ser resolvido exclusivamente por métodos exatos, em instâncias de grande porte.

Desta forma, técnicas aproximadas ou heurísticas tendem a ser alternativas viáveis para a sua solução. Neste capítulo, são propostas diferentes heurísticas com o intuito de obter bons limites inferiores para um valor ótimo do PETRRD.

Em princípio, para resolver este problema, pode-se utilizar um critério de priorizar tarefas muito simples: seu custo-beneficio. Ou seja, a tarefa que tiver menor custo de ativação e maior lucro é uma forte candidata a ser escolhida para fazer parte da solução, desde que atenda às restrições do problema. Trabalhar com a idéia de custo e benefício é muito utilizado em vários outros tipos de problemas como pode-se ver em [16, 17].

3.1 Heurística construtiva ADD

Esta heurística gulosa, a cada instante t do horizonte de tempo da instância em questão, gera uma lista das tarefas que estão disponíveis e que têm custo menor ou igual ao total

de recursos disponíveis para este instante. Esta lista é ordenada de forma crescente pela relação custo-benefício ($\frac{c_i}{l_i}$) das tarefas.

Seguindo a ordenação estabelecida, ativam-se as tarefas que puderem ser ativadas até que não haja mais recursos ou que não haja mais tarefas disponíveis. Vale enfatizar que, quando uma tarefa i é ativada, a quantidade de recursos é decrementada de c_i unidades e o lucro retornado é incrementado em l_i unidades, a partir do tempo seguinte. Assim, nem sempre todas as tarefas disponíveis num dado tempo poderão ser ativadas, devido à limitação de recursos em cada tempo. Após esta etapa, muda-se o instante de tempo em questão para o próximo valor $(t + 1)$ e repetem-se os procedimentos anteriores. O Algoritmo 1 mostra um pseudo-código do método ADD.

Algoritmo 1 ADD(inteiro T, inteiro recIni)

```
1: Q ← recIni;
2: L ← 0;
3: t ← 1;
4: while t ≤ T do
5:    Gera-se a lista D de tarefas disponíveis;
6:    Ordena-se D de acordo com c_i/l_i; //crescentemente
7:    i ← 1;
8:    while (Q>0)and(i ≤ |D|) do
9:       k ← D(i); //obtém o número (k) da i-ésima tarefa da lista D
10:      if c_k<Q then
11:         Ativa-se k;
12:         Q ← Q − c_k; //decrementa o total de recursos
13:         L ← L + l_k; //incrementa o lucro
14:      end if
15:      i ← i + 1;
16:   end while
17:   Q ← Q + L; //calcula os recursos disponíveis para o próximo instante
18:   t ← t + 1;
19: end while
20: return Q;
```

3.2 Heurística construtiva randomizada ADDR

Esta heurística é uma versão randomizada da ADD. Nesta versão, após a criação e ordenação da lista, em cada instante t, são disponibilizadas apenas as p% melhores tarefas (as p% tarefas de menor custo-benefício), criando assim uma LRC (lista restrita de candidatos). Por exemplo: se existem 8 tarefas para serem escolhidas e a LRC for composta

pelas 25% melhores tarefas, apenas as duas melhores tarefas farão parte da LRC. Entre elas, apenas uma será escolhida aleatoriamente. Se ela possuir custo menor ou igual à quantidade de recursos disponíveis, será ativada, caso contrário será removida da lista. A heurística prossegue até que não haja mais recursos ou até que não haja mais tarefas (da LRC) para serem ativadas. Passa-se então para o próximo instante de tempo e repete-se o processo. Este parâmetro p será chamado de alfa (α). Quanto menor o valor de α mais gulosa será a heurística e vice-versa.

3.3 Técnicas auxiliares

Algoritmos contrutivos do tipo ADDR normalmente não retornam boas soluções. Para tentar melhorar o desempenho deste construtivo básico, serão utilizadas algumas técnicas adicionais que visam impedir ou possibilitar a escolha de determinadas tarefas, como visto a seguir.

3.3.1 Eliminação de arcos

Por meio desta técnica, é possível eliminar alguns arcos, sem violar a precedência das tarefas envolvidas. Um arco (i,j) será removido se, após a remoção ainda for possível (sair de i e chegar em j, logicamente por outro caminho). Isto pode ser feito, pois se houver outro caminho que ligue i a j, a precedência explícita em (i,j) estará implícita por este caminho. Na Figura 3.1, é ilustrado um exemplo do uso desta técnica: o arco (1,3) poderá ser removido, pois existe outro caminho {(1,2) (2,3)} que mantém a precedência entre 1 e 3. De fato, para a ativação da tarefa 3, será necessária, além de 1, ativação também de 2. Esta técnica visa reduzir o espaço de busca de soluções do problema e conseqüentemente o tempo de execução de qualquer algoritmo.

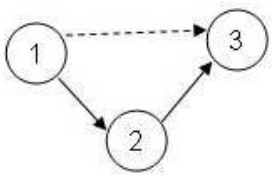


Figura 3.1: Exemplo de eliminação de arcos

3.3.2 Ponderação Prévia (PP)

Antes do início do algoritmo, estima-se qual o menor tempo de início de cada tarefa i ($inicio(i)$) da seguinte maneira: as tarefas sem predecessores têm tempo de início igual a 1; para as demais, o tempo de início é calculado como sendo o maior tempo de início dos predecessores diretos acrescido de uma unidade.

Desta forma, será suposto que as tarefas irão iniciar o quanto antes. Vale notar que isto é apenas uma estimativa, já que nada garante que a ativação das mesmas ocorra no menor tempo possível. Com a estimativa feita, estima-se também o lucro total da tarefa. Assim, a ordenação não será feita seguindo o critério usual ($\frac{c_i}{l_i}$), mas sim o custo c_i dividido pelo lucro total estimado. Esta técnica pode ser utilizada com qualquer uma das heurísticas construtivas propostas.

3.3.3 Fração de Corte

Esta é uma técnica utilizada na geração da lista de tarefas disponíveis. Neste caso, além dos critérios usuais, uma tarefa só será inserida na lista se retornar um lucro líquido não negativo, ou seja, se o lucro total da tarefa for maior ou igual ao seu custo. A fração de corte pode ser utilizada em qualquer instante de tempo t do horizonte de tempo em questão. Nos testes foram utilizados valores reais entre 0 e 1, ou seja, uma fração de corte de 0.3 indica que somente depois dos 30% primeiros instantes de tempo do horizonte de tempo será utilizada a fração de corte. Assim, o valor 0 (zero) indica que sempre será utilizada a fração de corte e o valor 1 (nunca) indica que nunca será utilizada.

A idéia por trás da fração de corte é não utilizar recursos com tarefas que não trarão lucro real (lucro líquido não negativo), disponibilizando os mesmos para outras tarefas mais atrativas.

3.3.4 Folga

A folga é uma relaxação da técnica de fração de corte. Nela, uma tarefa poderá constar na lista de tarefas disponíveis mesmo que não retorne lucro real, desde que o lucro total retornado por ela seja maior o igual ao seu custo dividido por um valor (a folga). Assim, um valor de folga de 1 indica que a tarefa deve retornar um lucro total maior ou igual

a seu custo, pois o custo será dividido por 1. Um valor de 2 indica que a tarefa deverá retornar um lucro total maior ou igual à metade de seu custo, pois o custo será dividido por 2.

Por exemplo: seja um valor de folga igual a 1,1; uma tarefa i com $c_i = 13$ e $l_i = 3$. Esta tarefa será ativada no tempo $t = 8$ num total de 11 unidades de tempo. O lucro total retornado por ela será de $((11 - 8) + 1) \times 3$, que é igual a 12. Desta forma, seu custo será 13 e o lucro total 12, o que, se for suposto que a fração de corte já esteja vigorando, não permitirá a ativação de i. No entanto, com a folga de 1,1, o custo será dividido por este valor, ficando assim 1,18, que é menor do que 12. Ou seja, ao utilizar a folga permite-se que i seja ativada.

Ao fazer isso, corrige-se, de certa forma, uma desvantagem da fração de corte: uma tarefa específica pode não trazer lucro real, mas pode disponibilizar tarefas (suas sucessoras) que o tragam. Assim, pode ser que valha a pena ativar uma tarefa pouco lucrativa, visando ativar suas sucessoras.

3.4 Buscas locais

Uma busca local é um procedimento que visa tentar melhorar a qualidade de uma solução já criada, por meio de pequenos refinamentos em sua estrutura. É uma técnica muito utilizada porque, geralmente, as soluções das heurísticas construtivas não representam uma solução ótimo local, principalmente aquelas que contém componentes aleatórios como a ADDR. Portanto, nestes casos, é recomendado o uso de métodos que investiguem soluções vizinhas a uma dada solução inicial.

As buscas locais mais comuns são aquelas que se utilizam de uma estrutura de vizinhança. A definição de vizinhança $V(s)$ de uma solução s é muito flexível e depende diretamente do problema e da representação da solução, no entanto, ela deve ser tal que permita sair de uma solução inicial s e percorrer elementos da vizinhança $V(s)$ em busca da melhor solução local.

No PETRRD, essa estrutura de vizinhança requer um cuidado muito especial, pois existem restrições que não poderão ser violadas. Isto faz com que o número de modificações (movimentos) possíveis seja reduzido. Preferiu-se utilizar refinamentos que tentem

remover, da solução atual, tarefas que possivelmente estejam consumindo recursos sem benefício à qualidade final da solução, tentando maximizar, sempre que possível, a quantidade de recursos disponível. Foram propostos quatro procedimentos de busca local que estão descritos a seguir.

3.4.1 Busca Local 1 (BL1)

Esta busca local visa eliminar, de uma solução corrente, as tarefas que não retornam lucro real, nem possibilitam a ativação de outras tarefas. A Figura 3.2 mostra um grafo que servirá de exemplo. As tarefas em branco representam as tarefas já incluídas na solução. Os números acima de cada tarefa indicam, respectivamente, o custo e o lucro da mesma.

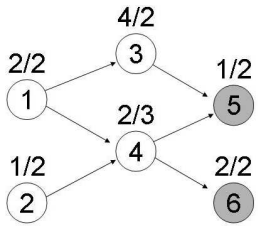


Figura 3.2: Exemplo da Busca Local 1

A tarefa 3 tem um custo de 4 e lucro de 2. Ela pôde fazer parte da solução porque a fração de corte e/ou a folga permitiram, visando que uma sucessora mais lucrativa (tarefa 5) fizesse parte da solução. Enfim, se a tarefa 3 for ativada no instante de tempo T, ela terá um lucro líquido de -2 (-4+2) causando, portanto, prejuízo ao objetivo final do problema.

Com a remoção desta tarefa 3, será eliminado também este prejuízo e melhorada em duas unidades a qualidade da solução. Vale notar que isto só poderá ser feito se a tarefa em questão não possuir sucessoras que façam parte da solução, para não inviabilizá-la. Por isto, este refinamento deverá começar pelas tarefas ativadas no último instante de tempo. Quando terminar esta etapa, serão analisadas, recursivamente, as que foram ativadas no penúltimo instante e assim sucessivamente. Desta forma se uma tarefa i for ativada no tempo t e uma sucessora j ativada em $t + 1$, ambas com lucro líquido negativo, primeiro será removida a tarefa j e posteriormente a tarefa i, mantendo, desta forma, a solução corrente sempre viável.

3.4.2 Busca Local 2 (BL2)

Esta busca local é uma extensão da Busca Local 1, com a diferença de analisar o lucro líquido de uma parte do grafo. Ou seja, o conjunto de tarefas sucessoras de uma tarefa i será analisado e dependendo do lucro líquido trazido por todo o conjunto, poderá ser eliminado ou não. Neste caso deve-se ter o cuidado de calcular o lucro total do subconjunto, levando em conta o menor tempo de ativação das tarefas envolvidas. A Figura 3.3 ajudará a entender esta decisão.

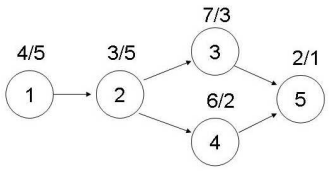


Figura 3.3: Exemplo da Busca Local 2

Suponha que a tarefa 5 seja ativada no último instante de tempo (lucro líquido de -1), as tarefas 3 e 4 ativadas no penúltimo instante (lucros líquidos de -1 e -2 respectivamente). As tarefas 3 e 4 mesmo trazendo lucro por dois instantes (lucro ao final do penúltimo e do último instante) de tempo não conseguem suprir os recursos gastos com suas ativações. No total, as tarefas 3, 4 e 5 têm custo de 15 e retornam lucro de 11, ficando com um prejuízo de 4 unidades.

Com a Busca Local 2, que percorre todas as tarefas, verificando suas sucessoras, este subconjunto de tarefas (sucessoras de 2) será removido, melhorando a qualidade da solução.

3.4.3 Busca Local 3 (BL3)

Esta busca local difere um pouco das anteriores porque não remove tarefas da solução, mas tenta antecipar o tempo de ativação das mesmas. Logicamente, se uma tarefa i (com lucro de 3, por exemplo) puder ser ativada num instante anterior ao que é atualmente ativado, ela retornará um lucro líquido maior (3 unidades a mais a cada unidade de tempo antecipada).

No entanto, esta busca local só se torna proveitosa se, por algum motivo, for possível liberar recursos no instante anterior da ativação de i, isto porque a heurística construtiva

sempre tentará ativar a tarefa o mais cedo possível, só não o fazendo pela falta de recursos.

Desta forma, há a sugestão de que esta busca local seja utilizada somente após as outras duas buscas locais BL1 e BL2. Já que estas ao (possivelmente) remover algumas tarefas, estariam liberando recursos que possibilitariam a antecipação de outras tarefas. Ou seja, a BL3 só deve ser usada quando a BL1 ou a BL2 consegue remover alguma tarefa.

Uma forma interessante de saber até que intervalo de tempo uma tarefa pode ser antecipada é tentar antecipar (em um intervalo de tempo) primeiramente as tarefas que foram ativas por último. Assim, elas farão parte do conjunto de tarefas que foram ativadas no penúltimo instante de tempo. Prossegue-se o algoritmo com estas tarefas (tentando antecipá-las) até que se chegará ao instante de tempo onde cada tarefa não poderá mais ser antecipada (pela falta de recursos ou pelas restrições de precedência).

3.4.4 Busca Local 4 (BL4)

Esta busca local utiliza o algoritmo de recriação que será explicado mais a frente, na seção 3.5.7. Este algoritmo, basicamente, fixa uma parte da solução atual e reconstrói o restante da solução por outro critério (diferente daquele usado pela heurística randomizada ADDR). São feitas dez iterações nesta busca local, sendo que cada uma é composta de uma recriação e de uma busca local (BL1). Estas várias iterações são necessárias porque, como será explicado, a recriação procura escolher tarefas de maneira menos gulosa que a heurística construtiva ADDR. Desta forma deve-se considerar a probabilidade da recriação não construir uma boa solução e, portanto, utilizando-se mais de uma iteração, espera-se que a solução resultante seja de melhor qualidade.

3.5 Algoritmos Evolutivos

Neste trabalho, o objetivo principal é desenvolver módulos que permitam gerar heurísticas eficientes na solução do PETRRD. Para tanto, além de propor métodos construtivos e de busca local, apresenta-se também o que é aqui denominado como "Técnicas auxiliares" compostas por etapas de pré-processamento (eliminação de arcos e ponderação prévia), fração de corte e folga.

De posse destes procedimentos, passou-se à etapa de inserí-los em duas meta-heurísticas com diferentes filosofias. Uma delas está baseada no conceito de heurística de população [18] ou Algoritmos Evolutivos (AEs). A segunda meta-heurística foi escolhida dentre aquelas que trabalham com um método construtivo e um ou mais métodos de busca local tais como GRASP, VNS, Busca Tabu entre outros.

Destes foi selecionada a estrutura do tipo GRASP devido a sua simplicidade, a sua semelhança com os AEs aqui propostos nas etapas de inicialização e busca local e pela sua eficiência mostrada em diferentes aplicações da área de otimização combinatória [19].

Um paradigma muito utilizado, ao se tratar de meta-heurísticas, é o de trabalhar simultaneamente com um conjunto de soluções que servem como uma base de dados para a geração de novas soluções. As heurísticas denominadas Algoritmos Evolutivos (AEs) e, em particular, o seu representante mais popular, os Algoritmos Genéticos (AGs) trabalham fundamentalmente desta forma. O AG proposto originalmente por Holland [20], tem como motivação a teoria da evolução das espécies de Charles Darwin. Nesta meta-heurística, inicialmente, são geradas algumas soluções (indivíduos) que formam o que se chamará de população inicial. A partir de então, estas soluções (ou indivíduos) são combinadas de acordo com algum mecanismo, formando assim um novo grupo de soluções chamadas de "filhos". No AG, os mecanismos clássicos de combinação são chamados de *crossover* e mutação.

Cada solução de uma população passa por um critério de seleção que definirá quais soluções farão parte de uma nova geração e quais serão eliminados. Geralmente, as soluções são escolhidas (ou eliminadas) de acordo com a sua qualidade (chamado de *fitness* ou aptidão).

Eventualmente, as soluções podem passar por um processo de perturbação aleatória (chamado de mutação), onde pretende-se alterá-las para que possam gerar soluções diversificadas.

Os critérios de parada para esta meta-heurística usualmente são o número total de gerações e/ou o número de gerações sem alteração da melhor solução ou outro critério referente à qualidade das soluções.

Nos algoritmos baseados em conceitos de AGs aqui propostos, são inseridos vários

procedimentos que normalmente não estão presentes em AGs tradicionais. Desta forma, os algoritmos propostos são denominados Algoritmos Evolutivos (AEs).

Será explicado a seguir como foi feita a modelagem do Algoritmo Evolutivo proposto e como são os procedimentos utilizados na sua execução.

Vários trabalhos sobre Algoritmos Genéticos, e mais geralmente sobre AEs, estão disponíveis na literatura, explicando melhor estas e muitas outras técnicas utilizadas. Para uma leitura mais aprofundada sobre o tema sugere-se ler [21, 22, 23, 24, 25].

3.5.1 Representação da solução e função de aptidão

Será utilizado um vetor de números inteiros para representar uma solução do problema proposto. Nesta modelagem, cada índice do vetor representa uma tarefa do grafo de entrada e cada valor do vetor indica o tempo (do horizonte de tempo) no qual a tarefa será ativada. Por exemplo, na figura 3.4, o índice 3 (tarefa 3 do grafo) tem valor 4 (a tarefa 3 será ativada no instante 4 do horizonte de tempo). Quando uma tarefa não fizer parte da solução, pode-se colocar o valor $M > T$ onde T é o último instante de tempo do periodo considerado.

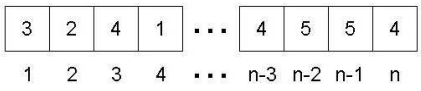


Figura 3.4: Exemplo de solução

Para avaliar uma solução, primeiramente, deve-se verificar sua viabilidade, isto é, se ela satisfaz as restrições do problema. Uma vez satisfeitas, calcular a qualidade da mesma é bem simples. A seguir (Algoritmo 2) é apresentado um pseudo-código que retorna a qualidade da solução. Para cada instante de tempo, verifica-se qual a soma dos custos (SC, linha 5) das tarefas ativadas neste instante e a soma do lucro (SL, linha 6) obtido pelas mesmas. O lucro atual é somado com SL (linha 7). Da quantidade de recursos disponíveis é subtraído SC e somado ao lucro atual (linha 8), para saber a quantidade recursos que estará disponível no instante seguinte.

Algoritmo 2 Avalia(solução S, inteiro recIni, inteiro T)

```
1: Q ← recIni;
2: L ← 0;
3: t ← 1;
4: while t ≤ T do
5:    SC ← custoTarefas(S, t);
6:    SL ← lucroTarefas(S, t);
7:    L ← L + SL;
8:    Q ← Q − SC + L;
9:    t ← t + 1;
10: end while
11: return Q;
```

3.5.2 Geração da população inicial

Neste trabalho, optou-se por utilizar uma população de tamanho fixo. Nas várias versões de AEs propostas, as populações têm tamanhos distintos, mas sempre fixos (dentro de cada versão). A população inicial é gerada pela heurística randomizada ADDR em todas as versões.

Vale notar que, em uma mesma população, não é interessante a presença de dois ou mais indivíduos iguais. Também é importante salientar que, nos AEs propostos, sempre que um indivíduo for gerado, passará por pelo menos uma das buscas locais citadas neste capítulo.

Para que estes indivíduos sejam gerados, no entanto, é preciso fornecer para a heurística ADDR o valor de alguns parâmetros, vistos na seção 3.3, tais como: fator de aleatoriedade α, fração de corte, folga e a utilização ou não de ponderação prévia.

Entre esses parâmetros, sem dúvida, o mais importante é o α que determinará quão guloso ou quão aleatório será o comportamento da heurística. Porém, uma boa calibração da fração de corte pode também ajudar muito a melhorar a qualidade da solução corrente.

De maneira geral, é muito útil que a calibração destes parâmetros seja feita com informações obtidas da instância que está sendo tratada, pois, como será visto no Capítulo 4, existem muitas configurações de instâncias que podem ter parâmetros ideais bem distintos. Serão explicadas, a seguir, as estratégias de calibração dos parâmetros propostas neste trabalho.

3.5.2.1 Calibração estatística (Calib1)

Nesta estratégia, utiliza-se uma amostragem de soluções, gerada até o momento pelo algoritmo, para definir qual o melhor valor para cada parâmetro. São realizadas preliminarmente 20 execuções para cada valor em cada um dos parâmetros. A ordem de testes dos parâmetros será: α, fração de corte, folga, ponderação prévia.

O parâmetro α será testado com valores de 0,05 até 0,4, em incrementos de 0,05. Cada um destes valores será testado 20 vezes (os outros parâmetros estarão fixos, nestes testes), sendo escolhido o valor de α que conseguir obter a solução de melhor qualidade.

De posse do melhor α, os parâmetros de fração de corte (valores de 0,2 até 1,0, em incrementos de 0,1) e folga (valores de 1,0 até 1,4, em incrementos de 0,1) também serão escolhidos da mesma maneira.

A utilização de ponderação prévia só apresenta dois valores possíveis ("sim", utilizar a ponderação ou "não", não utilizá-la). Também serão feitas 20 execuções com cada valor e a escolha do mesmo será de forma idêntica à dos demais parâmetros.

3.5.2.2 Calibração por densidade (Calib2)

É uma estratégia de calibração muito semelhante à calibração estatística, com exceção do parâmetro α. Este será definido de acordo com a densidade de arcos do grafo. Esta densidade pode ser definida como o número de arcos presentes no grafo dividido pelo total de arcos existentes se este grafo fosse completo.

Quanto menor a densidade, menor será o valor de α e vice-versa. Isto porque quanto menor o número de arcos, teoricamente, será maior o número de tarefas disponíveis a cada tempo, pois haverá menor número de precedências. Assim, para tentar garantir boas escolhas, dentre as tarefas disponíveis, será necessário reduzir o α, tornando a escolha mais gulosa. No entanto, esta densidade precisa passar por uma multiplicação com um fator de correção, pois dependendo do tipo de instância, este valor pode ser muito pequeno. No caso das instâncias A (ver capítulo 4 sobre as instâncias) este fator de correção é 10 (dez).

3.5.2.3 Calibração por densidade ponderada (Calib3)

Calibração semelhante à anterior. No entanto, o valor de α, neste caso, será a média ponderada da densidade do grafo e da porcentagem de tarefas inicialmente livres no mesmo. Esta porcentagem é a relação do número de tarefas inicialmente livres com o número total de tarefas.

Os pesos aplicados também podem variar de acordo com a instância em questão. Nas instâncias A, estes pesos foram de 9 (nove) para a densidade e 1 (um) para a porcentagem de tarefas inicialmente livres.

3.5.2.4 Calibração aleatória (Calib4)

Nesta estratégia, não existe uma calibração propriamente dita, porque cada vez que a heurística ADDR for executada, serão passados para ela valores de parâmetros escolhidos aleatoriamente.

No entanto, os valores de cada parâmetro estão restritos a um intervalo: α:]0 ; 0,3]; fração de corte: [0,3 ; 1,0]; folga: [1,0 ; 1,3]; ponderação prévia - "sim" ou "não". Um valor qualquer (com duas casas decimais) dentro de cada faixa será escolhido, com probabilidade uniforme.

3.5.3 Combinação de Soluções

A combinação de soluções (indivíduos) nos AEs tem como meta gerar novas soluções. O método mais comum em AGs é o chamado de cruzamento em ponto. Neste cruzamento, as soluções pais S_1 e S_2 são divididas em duas partes. A primeira parte de S_1 é então unida à segunda parte de S_2 e vice-versa. Este operador, em muitos casos, no entanto, gera soluções inviáveis.

Propõe-se portanto, combinar as duas soluções pais de outra forma: para cada tarefa, verifica-se qual é o tempo de início desta em cada pai e escolhe-se o menor deles. Se esta escolha causar a inviabilidade da solução, escolhe-se o valor presente no outro pai. Se mesmo assim a solução se tornar inviável, então a tarefa em questão não estará presente na solução filho. Este algoritmo de combinação será chamado de Comb1.

Outro algoritmo, mais específico para o problema proposto, consiste um criar uma lista única de tarefas que estejam disponíveis em cada pai, para cada instante de tempo. A partir dessa lista, serão escolhidas as tarefas que farão parte dos filhos. A seguir (Algoritmo 3), é apresentado o pseudo-código deste algoritmo que será chamado de Comb2.

Algoritmo 3 Comb2 (solução S_1, solução S_2, inteiro recIni, inteiro T)

```
1: Q ← recIni;
2: L ← 0;
3: Cria a Lista D (vazia);
4: Insere em D as tarefas disponíveis de S1 e S2;
5: t ← 1;
6: while t ≤ T do
7:    Escolhe as tarefas de D que farão parte de S;
8:    Coloca as tarefas em S, desde que não violem as restrições;
9:    Atualiza L e Q;
10:   Atualiza D;
11:   t ← t + 1;
12: end while
13: return S;
```

A escolha das tarefas que ocorre na linha 7 é aleatória. Isto porque a escolha das tarefas presentes nos pais S_1 e S_2 já foi feita por critério guloso e, assim, uma escolha aleatória neste momento poderia atenuar este critério, possibilitando que o "filho" tenha características levemente diferentes dos pais, esperando, neste caso, escapar de eventuais ótimos locais ainda distantes de um ótimo global.

3.5.4 Intensificação

O procedimento de mutação não é aplicado nos AEs propostos. Mas sugere-se para este problema a aplicação das buscas locais propostas sobre um mesmo indivíduo da população, na seguinte ordem: BL2 e BL3. Embora as buscas locais, como foram propostas, sempre mantenham ou aumentem (melhorem) a qualidade da solução, aplicá-las sobre todos os indivíduos consumiria um tempo computacional muito grande. Além do mais, se durante a geração da população inicial, os indivíduos já passarem pela BL2, esta não será aplicada novamente quando ocorrer uma intensificação.

A estratégia adotada foi utilizar uma seleção probabilística, ou seja, cada indivíduo tem uma probabilidade de realizar a intensificação proposta (BL2 e BL3) independente dos demais indivíduos. Foram propostas três estratégias probabilísticas:

- Int1: cada indivíduo tem probabilidade de 10%.
- Int2: cada indivíduo tem probabilidade de 40%.
- Int3: o melhor indivíduo tem probabilidade de 20%, o segundo melhor tem probabilidade de 18%, e assim sucessivamente até o décimo melhor indivíduo com probabilidade de 2% apenas.

3.5.5 Nova Geração

Para definir uma nova geração nos AEs propostos, a população corrente realiza combinações entre seus elementos dando origem a uma nova população. Estas duas populações terão, em todas as versões propostas, o mesmo tamanho. Para saber quais destes indivíduos continuarão na geração seguinte, utilizou-se o critério elitista, ou seja, dos n pais e n filhos sobrevivem os n melhores indivíduos distintos, onde n é o tamanho da população.

Resta definir como os pais serão escolhidos para combinar e gerar novos filhos. Utilizou-se uma estratégia de divisão da população em classes de indivíduos: os 20% melhores indivíduos farão parte da classe A da população, os 20% piores indivíduos farão parte da classe C e os demais farão parte da classe B.

A primeira estratégia adotada será escolher apenas indivíduos das classes A e B. Esta estratégia (A∪B) permite, inclusive, que dois indivíduos da mesma classe se combinem. A outra estratégia (AxB) obriga que um pai seja da classe A e o outro da classe B.

Da mesma forma que na geração da população inicial, quando um filho for gerado, este também passará por pelo menos duas buscas locais antes de fazer parte da população seguinte. Assim, as buscas locais serão aplicadas em três momentos: quando a população inicial for gerada, quando ocorrer uma intensificação (de acordo com as probabilidades citadas) e quando um filho for gerado.

3.5.6 Perturbações

É comum que, de tempos em tempos, o AG acabe convergindo para uma situação onde não seja fácil aprimorar a qualidade do melhor indivíduo da população corrente. Porém, não é simples perceber quando esta situação acontece. Desta forma, foram propostas

algumas estratégias para tratar este problema. Todas elas acontecem a cada 4 gerações consecutivas sem atualizar o melhor indivíduo gerado até o momento.

- P0: não faz absolutamente nada para sair desta situação.
- P1: retira 5 indivíduos, escolhidos aleatoriamente, e gera novos 5 para ocupar o lugar (da mesma forma que na geração da população inicial).
- P2: retira os 5 piores indivíduos e os substitui por novos (como na população inicial).
- P3: retira todos os indivíduos e gera uma nova população (como na população inicial).
- P4: alterna P3 com o algoritmo de recriação R1 (explicado a seguir).
- P5: alterna P3 com o algoritmo de recriação R2 (também explicado a seguir).

3.5.7 Recriação

A recriação aqui proposta pode ser visto como um processo de busca local, no entanto, ela mantém uma parte da solução analisada intacta e tentará reconstruir o restante da solução segundo algum critério.

Esta técnica pode chegar a resultados muito bons quando algumas partes da solução são independentes entre si, ou seja, quando o valor presente num segmento da solução não interfere nos demais segmentos. Quando isto acontece, pode-se fixar os segmentos responsáveis por boa parte da qualidade da solução e deixar os demais segmentos para serem recriados de forma mais livre do que a heurística construtiva utilizada na população inicial.

No entanto, no PETRRD esta independência raramente se verifica porque se uma tarefa possui tempo de ativação igual a 3, por exemplo, todas as tarefas sucessoras deverão apresentar tempo de ativação maior do que 3. Assim sendo, fixar um segmento da solução (pensando apenas nas tarefas) e tentar reconstruir o restante pode ser muito oneroso computacionalmente e nem sempre poderá chegar a melhorias desejadas.

Entretanto, existe uma estratégia simples que possibilitará o uso desta técnica de recriação: fixando um segmento de solução baseado não nas tarefas, mas baseado nos

seus tempos de ativação. Esclarecendo: as tarefas que forem ativadas até determinado tempo T_L estarão fixas. Sempre haverá, portanto, uma solução parcial válida contando apenas com as tarefas fixas. Esta solução parcial, porém, terá qualidade igual ou pior que a original. A saída é tentar ativar mais tarefas seguindo agora um critério que seja diferente do utilizado pela heurística construtiva ADDR, senão poderá ser recriada a mesma solução ou outra que seja muito semelhante.

Dois critérios foram propostos: o primeiro (R1) escolhe, aleatoriamente com probabilidade uniforme, algumas tarefas (dentre aquelas que estiverem livres) para fazerem parte da solução parcial. Esta escolha não poderá, logicamente, violar as restrições do problema e será feita seqüencialmente para cada instante de tempo a partir de T_{L+1} até T (limite do horizonte de tempo).

O segundo critério (R2) fará as escolhas da mesma forma, com exceção da probabilidade de cada tarefa que passará a ser calculada de acordo com o lucro que cada uma retorna. Será utilizada a técnica da roleta: cada tarefa terá uma probabilidade proporcional ao seu lucro. Por exemplo: se existirem três tarefas disponíveis com lucros de 4, 2, e 2 respectivamente, a primeira tarefa terá probabilidade de 50% (4/8), a segunda de 25% (2/8) e terceira também de 25% (2/8).

A Figura 3.5 mostra um esquema com as etapas da recriação. Em vermelho estão as tarefas que foram fixadas e em azul as que foram reconstruídas. Neste exemplo, o valor de T_L é 2.

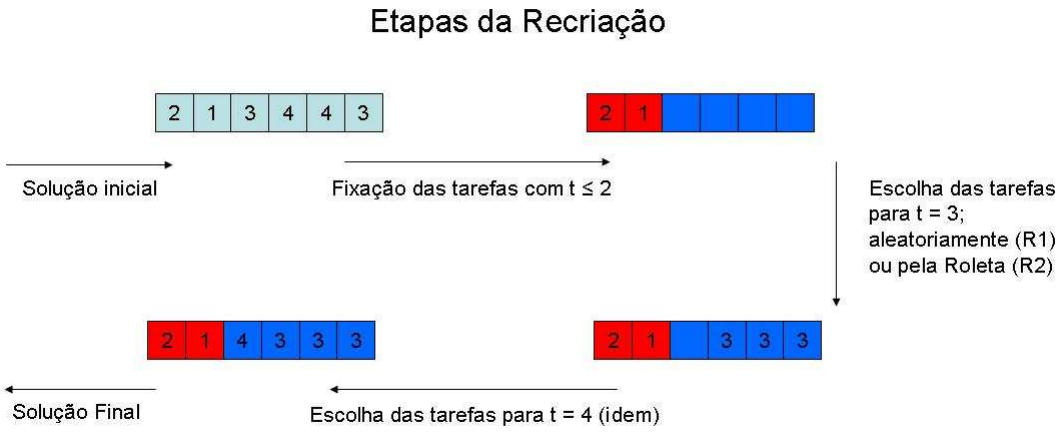


Figura 3.5: Esquema das etapas da recriação

3.5.8 Os algoritmos evolutivos propostos

Com todas essas opções de parâmetros propostos para serem utilizados, seria possível criar dezenas de versões de algoritmos evolutivos distintos. No entanto, foram propostos apenas seis versões para serem analisadas. A Tabela 3.1 mostra como é a configuração de cada uma das versões. A coluna Ver. indica a versão do AE em questão; a coluna Tam. o tamanho da população; Gers., o número de gerações; Calib., o tipo de calibração dos parâmetros; BLIni. indica quais buscas locais serão utilizadas no indivíduo, assim que ele for gerado; o algoritmo de perturbação utilizado será indicado por Pert.; a estratégia utilizada durante a combinação será indicada por EComb.; AComb. indicará o algoritmo utilizado na combinação; BLComb. indicará as buscas locais utilizadas em um indivíduo, após a sua geração pelos algoritmos de combinação e Int., por fim, indicará o tipo de intensificação utilizada.

Ver.	Tam.	Gers.	Calib.	BLIni.	Pert.	EComb.	AComb.	BLComb.	Int.
AE1	100	20	Calib2	1	P0	A∪B	Comb1	1,2,3	Int1
AE2	100	20	Calib2	1,2	P1	A∪B	Comb2	1,2,3	Int2
AE3	30	30	Calib4	1,2	P2	A∪B	Comb1	1,2,3	Int2
AE4	20	40	Calib1	1	P3	AxB	Comb2	1,2	Int3
AE5	20	40	Calib1	1	P4	AxB	Comb2	1,2	Int3
AE6	20	40	Calib1	1	P5	AxB	Comb2	1,2	Int3

Tabela 3.1: Configurações dos Algoritmos Evolutivos propostos

3.6 GRASP

Outro paradigma bastante diferente do utilizado pelos AEs é o que se pode encontrar na meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*). Considerada uma meta-heurística do tipo *restart*, o GRASP procura, a cada iteração, construir uma nova solução para o problema (etapa de construção) e aprimorá-la o máximo possível (etapa de busca local).

A etapa de construção de uma solução apresenta algumas características interessantes: por ser feita em etapas (a inclusão de elementos na solução é feita com um elemento por vez), ela é considerada adaptativa, ou seja, a escolha do k-ésimo elemento de uma solução é influenciada pelas escolhas dos elementos anteriores. Ela também é considerada gulosa, pois, a cada etapa, apenas alguns elementos (os melhores) farão parte de uma lista restrita

de candidatos (LRC) e somente eles poderão ser escolhidos para fazer parte da solução. No entanto, esta escolha é feita de forma aleatória, garantindo também um caráter randômico a esta construção. Esta escolha aleatória nos conjuntos LRC permite que a cada iteração do GRASP, uma solução diferente seja gerada pelo algoritmo construtivo.

O GRASP, que foi proposto originalmente por Feo e Resende [26], pode também contar com estratégias mais elaboradas, através de análises iniciais dos parâmetros utilizados e incluindo módulos adicionais como, por exemplo, combinar soluções de boa qualidade através de técnicas com a Reconexão por Caminhos [27], muito utilizada em várias versões de GRASP na literatura.

Da mesma forma que os AEs, o GRASP é uma meta-heurística muito estudada e difundida, principalmente, pela simplicidade de sua estrutura (as etapas de construção e busca local), aliada a sua competitividade na solução de diferentes problemas de otimização combinatória. Para uma leitura mais aprofundada sobre o assunto, pode-se recomendar a leitura de [19, 26, 27, 28, 29].

3.6.1 As versões de GRASP propostas

De forma semelhante aos AEs, foram propostas diferentes versões de GRASP, para se analisar, posteriormente no Capítulo 5. São apresentadas, na tabela 3.2, essas estruturas.

Versão	Calib.	BLs.	Extras
GR1	Calib1	BL1	-
GR2	Calib2	BL1	-
GR3	Calib3	BL1	-
GR4	Calib2	BL1,BL2,BL3	-
GR5	Calib2	BL1	Pool
GR6	Calib4	BL1,BL2	-
GR7	Calib2	BL1,BL2,BL4	-

Tabela 3.2: Configurações dos GRASPs propostos

3.6.2 Pool de soluções

Esta técnica funciona de maneira bem semelhante a uma população para o AE. Este *pool* nada mais é do que um conjunto de melhores soluções distintas, geradas até um dado momento numa execução do GRASP, que são reservadas para um processamento após

finalizar todas as iterações do mesmo.

O *pool* utilizado pela versão GR5 é criado inicialmente vazio. A cada iteração do GR5, a solução gerada é inserida no *pool* até que haja 20 soluções nele. A partir daí, uma solução só será inserida no *pool* se ela tiver qualidade melhor do que a pior solução presente no *pool*. Caso isto ocorra, a pior solução do *pool* sai e a solução em questão (do GRASP) entra no *pool*.

Assim, após todas as iterações do GRASP, haverá 20 soluções no *pool* que passarão pelo seguinte processo: a cada uma delas será aplicada a BL2 e, posteriormente, cada uma será combinada, através do algoritmo Comb1 com todas as demais uma única vez. Após as combinações serão aplicadas as buscas locais BL1, BL2 e BL3 nas soluções geradas pelo *pool*. Se alguma destas novas soluções tiver melhor qualidade do que as soluções geradas nas iterações do GRASP, esta solução será considerada a melhor gerada pelo GR5.

Capítulo 4

Instâncias para o PETRRD

Por se tratar de uma modelagem nova, proposta neste trabalho, não estão disponíveis instâncias do PETRRD que possam ser usadas como *benchmark* dos algoritmos propostos. Portanto, foram geradas várias instâncias com diferentes modelos de GADs para verificar o desempenho das heurísticas propostas. A seguir, é explicado como são geradas estas instâncias.

A - Nestas instâncias, 10% das tarefas não têm predecessor. As outras têm exatamente entre 1 (um) e 5 (cinco) predecessores escolhidos aleatoriamente.

B - Nestas instâncias a tarefa 1 não têm predecessor. A partir da tarefa 2, uma tarefa i tem uma probabilidade de 20% de ser sucessora de cada tarefa anterior a ela (desde a primeira até $(i-1)$ésima).

E - São instâncias onde não há qualquer forma de precedências, ou seja, as tarefas não têm predecessores. Este caso particular pode ser visto como um *multiple knapsack problem*.

Em todas as instâncias, os custos das tarefas variam de 1 a 50, enquanto os lucros variam de 1 a 10. Os valores são definidos aleatoriamente, com probabilidade uniforme.

Para as instâncias com até 1000 tarefas, o horizonte de tempo é limitado em $\sqrt{n}$ unidades. Para as demais instâncias, o limite é de $\sqrt[3]{n}$, sendo n o número de tarefas da instância em questão. Para identificar as instâncias, será utilizada a seguinte convenção: $XXXt$, onde XXX indicará o número de tarefas da instância e t a classe da mesma.

Por meio das instâncias E, pode ser feita a comparação deste modelo proposto com outro muito conhecido da literatura: *multiple knapsack problem* - *MKP*[15]. Neste pro-

blema, há, além dos objetos com pesos (ou tamanhos) e valores, M mochilas onde podem ser colocados os objetos. O objetivo é colocar os objetos nas M mochilas de forma a maximizar a soma dos valores obtidos.

Supondo que cada objeto seja uma tarefa (seu peso seja o custo, seu valor seja o lucro) e cada uma das M mochilas sejam associadas às várias unidades de tempo do horizonte em questão, a capacidade das mochilas seria a quantidade de recursos disponíveis em cada unidade de tempo. Logicamente que pequenas conversões aritméticas deveriam ser feitas para que houvesse a redução formal do PETRRD no MKP, no entanto deseja-se mostrar com isso apenas que o modelo PETRRD proposto é um problema de difícil solução, já que o MKP é classificado como NP-Completo [15].

Capítulo 5

Resultados Computacionais

Todos os testes foram executados em um computador pessoal Pentium4 3.0GHz HT, com 512Mbytes de memória RAM, sistema operacional Windows XP. O código fonte dos algoritmos foi escrito em linguagem C e compilado pelo C++ Builder 5.0.

5.1 Testes preliminares

Vários testes preliminares foram executados para que alguns valores dos diversos parâmetros utilizados pudessem ser calibrados para os testes mais completos que serão apresentados posteriormente. Entre os parâmetros mais importantes estão:

- As 10 iterações da BL4 (recriação): foram feitos testes preliminares com valores de 20 e 50 iterações, mas a qualidade das soluções obtidas com estes valores não justificou o tempo computacional gasto com estas iterações extras. Na verdade, na maioria dos casos, nem se verifica aumento significativo no valor das soluções (melhoria). No entanto, testes com 5 iterações mostraram que esta qualidade pode ficar comprometida em algumas instâncias.

- O valor de T_L para a recriação deve ficar entre 25% e 50% do horizonte de tempo. Valores escolhidos antes desta faixa, tendem a promover uma recriação muito aleatória e a qualidade da solução também pode ficar comprometida. Valores acima desta faixa, tendem a promover alterações mínimas na solução, uma vez que a maioria das tarefas é ativa até 50% do horizonte de tempo. Estas alterações mínimas acabam por não justificar o emprego desta busca local.

- O número de iterações para a calibração dos parâmetros alfa, fração de corte, folga e ponderação prévia também segue comportamento semelhante às iterações da BL4 (com mais iterações não há melhoras significativas, no entanto, com menos iterações a qualidade das soluções fica comprometida). O valor de 20 iterações (para cada parâmetro) foi escolhido por apresentar, nos testes preliminares, melhores resultados com um tempo computacional razoável.

- As faixas de valores para estes parâmetros (nas diversas técnicas de calibração) também foi definida com os valores que eram comumente escolhidos na maioria das instâncias. Esta faixa foi definida para evitar, portanto, que valores pouco comuns fossem testados, gastando assim menos tempo. Por exemplo: uma folga de 1,4 muito raramente era escolhida como sendo o melhor valor. Portanto, limitou-se a escolha da folga até o valor de 1,3. O mesmo (limitação dos valores) aconteceu com os demais parâmetros.

- O fator de correção da calibração por densidade (Calib2) depende bem mais da instância que está sendo estudada do que de outro fator qualquer. Assim, o valor 10 foi escolhido por ajustar melhor o valor de α na faixa de valores pré-definidos.

- Os pesos da Calib3 também foram testados de forma empírica e seguiram o mesmo objetivo do fator de correção: tentar ajustar melhor o α utilizado pela ADDR.

5.2 Testes Finais

Primeiramente, foi testada a utilização da Eliminação de Arcos. Como já foi dito, esta técnica remove do grafo de entrada alguns arcos redundantes, possibilitando, assim, que os algoritmos trabalhem com menos informações referentes ao grafo. A Tabela 5.1 mostra o tempo computacional gasto pela heurística GR1 em algumas instâncias dos tipos A e B. A coluna "n" indica o nome da instância testada. A coluna "Com" indica o tempo gasto (em segundos) utilizando-se a eliminação de arcos. A coluna "Sem" mostra o tempo gasto sem utilizar esta técnica. Pode-se perceber que na instâncias A, a utlização desta técnica torna a heurística um apenas pouco mais rápida. No entanto, nas instâncias do tipo B esta diferença é considerável a favor da utilização da mesma. Desta forma, em todos os experimentos realizados posteriormente será utilizada a Eliminação de Arcos.

n	Com	Sem
50a	0,7	0,7
50b	0,7	0,7
300b	28,8	34,2
500a	83,5	85,6
700b	231,4	284,4
1000a	406,5	401,3
1500a	395,8	435,9

Tabela 5.1: Comparação do tempo computacional gasto pela Eliminação de Arcos (em segundos)

Na Tabela 5.2, são apresentados os resultados médios obtidos pelas primeiras heurísticas nas instâncias A. A primeira coluna indica a instância testada. A partir da segunda, são mostrados os resultados das várias técnicas empregadas (ADD simples, ADD com ponderação prévia, ADD com fração de corte (valor = 0,7), ADD com fração de corte (0,7) e folga (1,1), ADD com busca local BL1). Em negrito, estão os melhores resultados para cada instância; os demais valores indicam a diferença percentual do valor em questão para o melhor valor obtido, ou seja, a distância desse valor para o melhor.

Nos testes realizados, a utilização de pelo menos uma das técnicas propostas ajudou, em todos os casos, a melhorar os resultados obtidos pela heurística gulosa ADD. Além disso, as melhoras mais significativas, em geral, foram obtidas pela utilização da busca local. Na instância 200a, por exemplo, o valor obtido pela ADD ficou 66,8% distante do valor obtido pela BL1: isto significa que a ADD só alcançou aproximadamente $\frac{1}{3}$ do valor da BL1. O tempo gasto pelas diversas técnicas, na Tabela 5.3, porém, não variou muito. Vale notar que em todos os experimentos realizados neste capítulo foi utilizada a Eliminação de Arcos.

Com relação ao tempo computacional gasto, verificou-se que o emprego das técnicas não chega a comprometer o desempenho do algoritmo construtivo, já que o acréscimo de tempo é pequeno. Mais um teste foi realizado agora com a primeira versão GRASP GR1 e uma versão básica chamada GR1p (GR1 puro). Esta versão é idêntica ao GR1 com a exceção de que o valor de fração de corte é fixo em 1 (ou seja, não há fração de corte), a folga fixa em 1 (não há folga) e a ponderação prévia em 0 (não há ponderação prévia). Os algoritmos foram executados três vezes (para cada instância) e a média do valor das soluções obtidas é mostrada na Tabela 5.4. A coluna % indica a distância que GR1p ficou

n	ADD	pp	frac	folga	BL1
50a	53,2%	53,2%	**47**	**47**	**47**
100a	54,9%	54,9%	20,2%	20,2%	**297**
150a	55,4%	55,4%	7,1%	7,1%	**574**
200a	66,8%	65,8%	8,7%	8,7%	**588**
250a	62,2%	64,5%	**987**	**987**	**987**
300a	61,6%	62,9%	7,0%	7,0%	**1684**
350a	48,5%	48,5%	3,4%	3,6%	**1859**
400a	27,6%	27,6%	**4975**	**4975**	0,2%
450a	26,4%	19,6%	**6424**	**6424**	5,5%
500a	20,3%	20,2%	**10793**	0,0%	4,1%
550a	32,7%	32,9%	0,2%	**7417**	2,9%
600a	33,0%	33,0%	1,2%	**6858**	7,3%
650a	31,0%	31,0%	0,1%	**11014**	3,6%
700a	11,7%	11,7%	**20473**	0,0%	2,5%
750a	2,6%	2,6%	**28355**	0,0%	1,5%
800a	2,2%	4,6%	**30480**	0,0%	1,2%
850a	7,9%	**31358**	1,1%	1,2%	3,1%
900a	9,2%	0,6%	0,2%	**24952**	1,9%
950a	0,2%	0,2%	0,0%	**59275**	0,2%
1000a	1,0%	**57946**	0,5%	0,5%	0,6%
1100a	56,7%	56,3%	7,7%	7,7%	**2391**
1200a	60,0%	59,6%	2,7%	2,8%	**2922**
1300a	59,3%	59,5%	**2703**	0,1%	0,6%
1400a	53,5%	52,7%	**2349**	**2349**	2,6%
1500a	56,9%	57,0%	0,1%	0,2%	**3645**
1600a	60,5%	60,2%	0,8%	0,8%	**3256**
1700a	60,2%	59,7%	2,2%	2,2%	**4416**
1800a	60,1%	59,5%	1,7%	1,7%	**3840**
1900a	57,8%	57,5%	**3881**	0,0%	1,6%
2000a	62,6%	62,6%	0,8%	0,9%	**5384**

Tabela 5.2: Tabela com os resultados das heurísticas determinísticas

de GR1.

A aplicação das técnicas auxiliares contribui para que GR1 obtivesse, na maioria dos testes, resultados melhores que GR1p. Em relação ao tempo computacional gasto, a versão GR1 consome, em média, 66% a mais do que a versão GR1p. Isto se deve, principalmente, às iterações utilizadas para a calibração dos parâmetros fração de corte, folga e ponderação prévia - que não acontece na versão GR1p. Desta forma, ao utilizarmos a Calib1 em GR1, estamos acrescentando mais 320 iterações (lembrando que são 20 iterações para cada valor de cada parâmetro: 9 valores de fração de corte; 5 valores de folga; 2 valores de ponderação prévia). Portanto, o tempo gasto apenas com as 500 iterações do GR1 não é tão maior

n	ADD	PP	frac	folga	BL1	n	ADD	PP	frac	folga	BL1
50a	0,5	0,8	0,5	0,5	0,5	800a	185,8	199,1	201,7	199,8	208,1
100a	1,7	2,2	1,7	1,7	1,9	850a	225,0	232,8	245,5	267,5	258,9
150a	3,8	4,7	3,9	3,8	4,2	900a	279,8	290,8	296,4	411,4	310,3
200a	7,8	9,4	7,8	7,8	8,6	950a	253,9	267,2	260,8	255,2	290,5
250a	13,6	15,6	13,8	13,4	15,3	1000a	301,7	318,4	341,7	310,5	348,0
300a	20,2	23,0	20,3	19,9	22,0	1100a	159,4	192,4	163,0	181,3	159,4
350a	30,5	33,9	30,6	30,8	33,1	1200a	207,8	248,6	220,0	208,4	209,4
400a	40,0	42,3	39,9	38,6	43,6	1300a	243,8	295,9	250,6	252,0	246,9
450a	51,6	55,2	52,4	51,1	56,3	1400a	281,2	339,8	288,6	291,3	293,8
500a	67,2	67,5	66,7	64,4	70,3	1500a	320,3	385,9	351,7	323,6	370,4
550a	82,8	87,7	86,3	83,3	92,2	1600a	400,0	480,6	424,7	399,1	434,3
600a	106,3	112,7	111,7	108,3	117,2	1700a	450,0	528,8	467,5	453,1	470,3
650a	125,0	132,0	132,2	127,5	139,5	1800a	506,2	589,9	526,7	505,8	520,3
700a	145,8	148,4	148,8	145,3	156,1	1900a	582,8	700,5	594,1	592,0	639,1
750a	156,6	162,5	168,3	162,3	174,2	2000a	681,2	784,7	665,9	674,9	689,0

Tabela 5.3: Tabela com tempos computacionais (em milisegundos) das heurísticas determinísticas

n	GR1p	GR1	n	GR1p	GR1
50a	0,0%	**47**	800a	0,5%	**30660**
100a	0,3%	**304**	850a	1,6%	**33135**
150a	0,0%	**575**	900a	1,9%	**26764**
200a	0,8%	**599**	950a	0,1%	**59590**
250a	1,6%	**1004**	1000a	0,0%	**60559**
300a	0,6%	**1711**	1100a	0,9%	**2464**
350a	1,5%	**1937**	1200a	1,6%	**2985**
400a	1,6%	**5168**	1300a	2,2%	**2762**
450a	4,8%	**6968**	1400a	2,7%	**2393**
500a	3,4%	**11266**	1500a	2,4%	**3743**
550a	6,5%	**8107**	1600a	0,5%	**3298**
600a	6,1%	**7177**	1700a	1,1%	**4467**
650a	3,4%	**11245**	1800a	1,6%	**3918**
700a	1,8%	**22131**	1900a	1,7%	**3926**
750a	0,4%	**28212**	2000a	3,8%	**5570**

Tabela 5.4: Comparação do GR1 com e sem as técnicas auxiliares

assim do que na versão GR1p.

Assim, o emprego das técnicas auxiliares propostas conseguem melhorar significativamente o desempenho tanto da heurística determinística ADD quanto das demais (meta)heurísticas GRASP em relação à qualidade das soluções geradas, sem onerar significativamente os tempos computacionais. O emprego de boas estratégias de calibração e buscas locais também ajudará muito a melhorar os resultados, como será visto mais a

frente. A partir daqui, serão analisadas as heurísticas propostas de acordo com as configurações apresentadas no Capítulo 3. Somente nas instâncias da classe E estas heurísticas serão novamente modificadas.

Na Tabela 5.5 é descrita a classificação obtida por cada versão dos AEs propostos no conjunto das instâncias A. Nesta tabela, cada célula mostra a classificação da metaheurística associada, decorrente da qualidade da solução obtida pela mesma. Assim, o valor 1 indica que a heurística obteve o melhor resultado para aquela instância.

A última linha da Tabela 5.5 mostra uma das formas de classificar os AEs. Nesta linha é descrita a soma de todos os valores obtidos por cada heurística. Neste sentido, o melhor desempenho foi do AE5, seguido de: AE6, AE4, AE2, AE1 e AE3.

Como o objetivo destes resultados preliminares, ilustrados na Tabela 5.5, é o de selecionar apenas alguns dos AEs para testes adicionais, para se certificar quais são realmente as melhores versões, foi efetuada uma segunda análise: classificar os AEs de acordo com o número de vezes em que este obteve um dos 3 melhores resultados (1, 2 ou 3). Por esta avaliação, a classificação, do melhor para o pior, ficou da seguinte forma: AE5 (22 melhores resultados), AE4 (21 melhores resultados), AE6 (21 melhores resultados), AE2 (16 melhores resultados), AE1 (9 melhores resultados) e AE3 (8 melhores resultados).

Os piores resultados em ambas as avaliações ficaram com AE1, AE2 e AE3. Embora as conclusões feitas empiricamente possam trazer equívocos, ao menos nos testes realizados se conclui que algumas decisões são desaconselháveis. Por exemplo: a estratégia de combinação A∪B ao invés de AxB no processo de seleção dos reprodutores (Tabela 3.1); o algoritmo de combinação Comb2 (lista única de tarefas dos dois pais) utilizado nas 4 versões com melhores resultados ao invés do Comb1 utilizado nas duas piores versões. De qualquer forma, pelas avaliações adotadas, selecionamos as versões AE4, AE5 e AE6 para futuros testes, descartando as versões AE1, AE2 e AE3.

Sobre as versões do GRASP, com os resultados mostrados na Tabela 5.6, também é possível fazer algumas considerações interessantes. O GR1, que utiliza valores pré-determinados de α (Calib1), embora possa fazer testes com estes valores, não conseguiu bons resultados. Isto se deve, possivelmente, ao fato de que os melhores valores de α, para certas instâncias, são bem diferentes dos valores pré-determinados. No entanto, se

n	AE1	AE2	AE3	AE4	AE5	AE6
50a	1	1	1	1	1	1
100a	5	5	1	1	1	1
150a	2	1	2	2	2	2
200a	5	3	6	2	4	1
250a	6	5	1	2	4	3
300a	6	5	1	3	4	2
350a	3	1	6	5	2	4
400a	4	6	3	5	1	2
450a	5	4	6	2	1	3
500a	5	4	6	1	2	3
550a	5	1	6	2	3	4
600a	4	5	6	2	3	1
650a	6	4	5	1	2	3
700a	5	4	6	3	1	2
750a	5	3	6	4	1	2
800a	3	4	6	5	2	1
850a	5	4	6	3	1	2
900a	6	3	5	4	1	2
950a	5	4	6	3	1	1
1000a	4	3	6	5	1	2
1100a	4	2	6	5	1	2
1200a	6	1	3	2	5	4
1300a	6	1	4	2	3	5
1400a	2	3	6	4	5	1
1500a	4	3	2	1	6	5
1600a	2	4	6	3	1	5
1700a	1	2	6	3	4	5
1800a	3	4	5	2	1	6
1900a	4	1	5	3	2	6
2000a	2	1	4	5	6	3
Total	124	92	138	86	72	84

Tabela 5.5: Classificação dos AEs para as instâncias da classe A

for permitido que a Calib1 teste mais valores, o tempo computacional gasto somente para estes testes comprometeria muito o desempenho da heurística. No caso dos AEs que utilizam esta Calib1 (AE4, AE5 e AE6), isto passa desapercebido exatamente porque as perturbações ajudam a gerar outras soluções, sem necessariamente, reutilizar este valor de α ruim. Portanto, parece ser uma opção melhor, no caso do GRASP, utilizar uma outra calibração.

Problema oposto ocorre com o GR3. Para tentar fugir dos valores pré-determinados, a Calib3, utilizada nesta versão, precisa ajustar dois fatores extras (o fator de correção

n	GR1	GR2	GR3	GR4	GR5	GR6	GR7
50a	1	1	1	1	1	1	1
100a	1	1	1	1	1	1	1
150a	2	5	5	4	5	2	1
200a	7	2	5	6	3	4	1
250a	1	7	1	1	5	1	6
300a	7	3	4	2	4	1	6
350a	3	6	7	2	5	1	4
400a	7	3	5	2	6	4	1
450a	6	5	1	2	4	7	3
500a	5	6	3	1	4	7	2
550a	4	6	5	2	3	7	1
600a	1	4	5	5	2	7	3
650a	7	4	6	3	2	5	1
700a	2	3	6	4	7	5	1
750a	7	2	5	3	6	4	1
800a	7	3	4	5	2	6	1
850a	6	4	5	2	3	7	1
900a	5	4	6	3	7	2	1
950a	3	5	4	2	7	6	1
1000a	5	6	2	7	4	3	1
1100a	4	1	6	7	2	5	3
1200a	5	4	1	2	3	7	5
1300a	3	5	1	2	6	4	7
1400a	2	5	3	1	6	7	4
1500a	1	5	6	4	3	2	7
1600a	5	3	4	7	2	1	5
1700a	3	2	7	5	4	1	6
1800a	7	5	6	3	2	4	1
1900a	7	4	6	5	3	2	1
2000a	7	1	6	5	3	4	2
Total	131	115	127	99	115	118	79

Tabela 5.6: Classificação dos GRASPs para as instâncias da classe A

e os pesos). Este ajuste não é trivial, uma vez que foram feitos testes experimentais apenas para se definir estes valores. Assim, mesmo que se consiga estabelecer um valor considerado melhor, este valor servirá bem para algumas instâncias e para outras talvez não. Este problema também ocorre na Calib2 (calibragem com fator de correção), no entanto, por se tratar de um fator apenas, fica menos difícil ajustá-lo.

Estas duas versões do GRASP (GR1 e GR3) serão deixados de lado por não apresentarem bons resultados, juntamente com o GR5 que, embora, tenha conseguido boas classificações apresentou um comportamento ruim: os melhores resultados apresentados

por ele não eram provenientes das operações realizadas no *pool*. Foram realizados novos testes alterando-se o algoritmo de combinação utilizado no *pool* para o Comb2 que apresentou, no caso dos AEs, desempenho melhor. O resultado foi que ocorreram pouquíssimas atualizações da melhor solução provenientes do *pool* e mesmo assim o resultado final foi, em geral, equivalente ao *pool* com o Comb1 (versão original). A retirada do módulo do *pool* desta versão do GRASP deixa a estrutura do algoritmo idêntica à do GR2 (ver Tabela 3.2). Portanto, optou-se por não continuar os testes com esta versão GR5.

Ficou-se, então, com 7 versões para prosseguir os testes: AE4, AE5, AE6, GR2, GR4, GR6 e GR7. A qualidade da solução obtida, somente entre elas, e o tempo computacional, em segundos, gasto por cada uma, nas mesmas instâncias A, podem ser vistos nas Tabelas 5.7 e 5.8, respectivamente. Na Tabela 5.7, os valores em negrito representam a melhor solução da instância e os demais valores representam a distância em percentual desta melhor solução.

Um resultado interessante pode ser visto se forem analisadas separadamente as instâncias de 700a até a 1000a, na Tabela 5.7. Nestas instâncias, a porcentagem de tarefas que são ativadas (do total de tarefas) é bem elevada (em torno de 90%) em comparação com as demais instâncias. Este foi um dos fatores que provocaram a alteração do cálculo do horizonte de tempo (para as instâncias com 1000 tarefas ou menos, o horizonte de tempo é igual à $\sqrt{n}$; para as instâncias maiores este horizonte é igual à $\sqrt[3]{n}$). Porém, a incorporação de muitas tarefas reduz um pouco a dificuldade, pois não será preciso se preocupar muito com quais tarefas serão ativadas e sim quando fazê-lo, apenas.

Explicando melhor: se existem recursos (e tempo) suficientes para ativar uma quantidade maior de tarefas, pouca diferença fará se uma tarefa i for ativada no tempo 14 ou 15, por exemplo. Isto porque, primeiro: a qualidade final da solução será um valor numericamente grande (quanto mais tarefas forem ativadas, mais lucro e, portanto, maior o valor da solução); segundo: as sucessoras de i serão, potencialmente, ativadas também, ou seja, não será impedida a ativação de outras tarefas.

Devido a isso, a BL4 consegue apresentar bom desempenho porque cada recriação é feita seguindo o critério da roleta para a escolha de quais tarefas entrarão na solução (ao invés do critério usual da heurística construtiva que é estritamente guloso). Portanto, a BL4 poderá percorrer um sub-espaço de soluções maior do que as demais estratégias.

n	GR2	GR4	GR6	GR7	AE4	AE5	AE6
50a	**47**	**47**	**47**	**47**	**47**	**47**	**47**
100a	**304**	**304**	**304**	**304**	**304**	**304**	**304**
150a	0,2%	0,2%	0,1%	**576**	0,1%	0,1%	0,1%
200a	0,2%	1,7%	0,9%	**619**	0,2%	0,5%	0,1%
250a	0,5%	**1004**	**1004**	0,4%	0,2%	0,5%	0,3%
300a	0,3%	0,1%	**1744**	0,5%	0,5%	0,7%	0,1%
350a	1,3%	0,6%	0,5%	0,8%	0,8%	**1951**	0,6%
400a	4,5%	4,5%	4,9%	**5482**	4,5%	1,8%	2,4%
450a	0,5%	**7053**	2,1%	0,2%	0,2%	0,1%	1,7%
500a	1,6%	0,7%	1,9%	0,9%	**11433**	0,4%	0,9%
550a	0,8%	0,3%	1,4%	**8151**	0,2%	0,5%	0,9%
600a	0,6%	0,8%	2,6%	**7166**	0,7%	0,9%	0,5%
650a	0,6%	0,6%	0,7%	0,3%	**11359**	0,0%	0,3%
700a	7,0%	8,4%	9,5%	**23770**	6,2%	2,7%	5,3%
750a	2,0%	2,2%	2,6%	**29208**	2,7%	2,0%	2,3%
800a	3,8%	3,9%	4,2%	**32011**	4,6%	3,3%	1,9%
850a	11,0%	10,9%	11,3%	**37315**	10,8%	5,5%	5,7%
900a	5,8%	5,8%	5,6%	**28431**	5,7%	2,0%	2,7%
950a	1,0%	0,8%	1,1%	**60166**	1,3%	0,7%	0,7%
1000a	1,0%	1,1%	0,9%	**61152**	1,0%	0,6%	0,7%
1100a	**2472**	0,8%	0,4%	0,2%	0,9%	0,3%	0,5%
1200a	0,3%	**3001**	0,7%	0,5%	0,8%	1,1%	1,1%
1300a	0,2%	**2763**	0,1%	0,8%	0,5%	0,5%	1,2%
1400a	1,1%	**2411**	1,5%	0,8%	1,7%	2,0%	0,9%
1500a	0,7%	0,5%	**3740**	0,8%	0,3%	1,7%	1,0%
1600a	0,1%	0,8%	**3318**	0,6%	0,4%	0,3%	0,9%
1700a	0,1%	0,4%	**4476**	0,6%	0,3%	0,5%	0,7%
1800a	0,7%	0,3%	0,6%	**3957**	1,1%	0,9%	2,0%
1900a	0,4%	0,4%	0,1%	**3956**	0,4%	0,4%	1,1%
2000a	**5629**	0,7%	0,5%	0,2%	1,6%	1,8%	1,2%

Tabela 5.7: Resultados médios obtidos pelos melhores AEs e GRASPs

Para comprovar que este comportamento se mantém com o aumento do tamanho das instâncias (e conseqüentemente com o aumento do horizonte de tempo), alterou-se o horizonte de tempo das instâncias 1100a, 1200a, 1300a, 1400a. Os valores antigos ($\sqrt[3]{n}$) passaram a ser $\sqrt{n}$ e o restante das informações da instância foram mantidos. Foram executadas as versões GRASPs sobre as novas instâncias e o resultado médio pode ser visto na Tabela 5.9.

No entanto, para se analisar a confiabilidade das meta-heurísticas no que se refere à qualidade das soluções geradas, realizaram-se mais algumas baterias de testes que serão descritas a seguir.

n	GR2	GR4	GR6	GR7	AE4	AE5	AE6
50a	0,6	1,1	0,7	3,8	1,7	1,7	1,1
100a	2,0	3,7	2,3	15,4	5,9	6,0	4,0
150a	5,4	8,6	5,5	34,9	12,9	13,1	8,7
200a	10,9	17,3	10,3	74,3	22,8	23,4	15,5
250a	18,9	29,0	18,8	125,1	43,8	39,7	26,9
300a	27,7	43,8	30,4	189,3	54,3	56,7	36,9
350a	43,2	64,9	44,2	287,6	82,0	90,0	58,0
400a	53,9	110,7	94,3	395,3	115,1	127,4	88,7
450a	71,7	179,3	161,2	621,7	181,5	190,7	134,4
500a	94,6	368,5	370,8	990,6	407,6	341,2	216,4
550a	119,5	312,1	285,1	1046,3	350,1	307,4	208,1
600a	145,1	319,2	277,5	1205,2	327,7	341,3	241,5
650a	175,1	319,4	489,8	1725,7	601,5	570,3	335,4
700a	194,2	358,1	1032,1	2414,5	748,4	838,8	691,3
750a	223,3	417,0	1791,9	4089,6	976,1	1588,6	1164,7
800a	262,8	494,5	1951,5	4698,5	1241,3	2190,8	1151,0
850a	250,5	479,7	2018,2	5120,0	1557,8	1966,6	1376,8
900a	353,9	567,9	1969,6	5525,4	1542,4	2091,6	1486,1
950a	363,8	666,2	4859,4	8443,9	2516,9	3777,1	2232,2
1000a	461,5	760,0	5425,3	7396,8	2529,5	4139,7	3360,1
1100a	203,6	312,9	266,8	1158,6	445,3	466,2	283,5
1200a	256,0	405,7	352,4	1538,1	591,4	609,2	375,9
1300a	319,3	446,8	369,2	1769,0	671,9	673,7	424,8
1400a	364,8	543,3	396,2	1896,3	781,2	776,7	477,3
1500a	371,0	651,5	694,4	2297,0	912,8	932,8	583,6
1600a	468,0	825,4	706,2	2411,5	1147,1	1176,9	732,3
1700a	555,6	901,7	865,0	2784,5	1317,1	1395,2	817,4
1800a	659,7	1031,7	869,9	3080,0	1492,8	1519,7	896,0
1900a	703,6	1346,4	887,4	3803,2	1951,3	2025,8	1194,3
2000a	624,2	1493,3	947,9	4734,0	2153,7	1917,4	1395,9

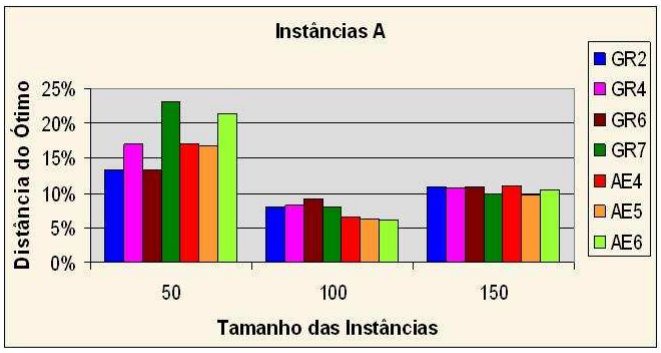
Tabela 5.8: Tempo computacional médio (em segundos) dos AEs e GRASPs

n	GR2	GR4	GR6	GR7
1100a*	3,1%	2,9%	3,2%	**83662**
1200a*	3,2%	3,2%	3,2%	**96630**
1300a*	4,5%	4,7%	4,7%	**102197**
1400a*	2,0%	2,0%	2,0%	**116827**

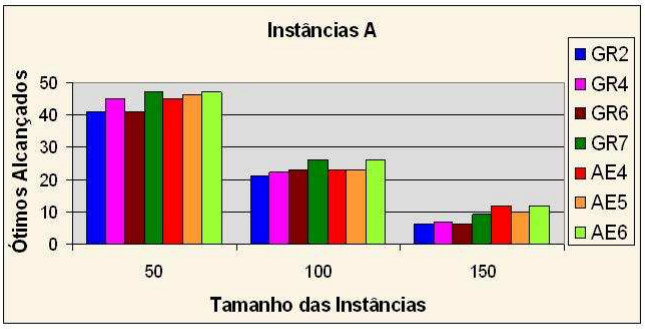
Tabela 5.9: Resultados médios para as instâncias modificadas

Nas instâncias com poucas tarefas (até 150, por exemplo), foi utilizada a formulação matemática proposta para obter ótimos globais através do *software GLPK*[30]. Foram geradas 50 novas instâncias, dos tipos A e B, de tamanhos 50, 100 e 150. Usando o GLPK, foi obtido o valor ótimo para cada uma delas e definiu-se este valor ótimo como sendo o

valor-alvo a ser alcançado por cada meta-heurística. As meta-heurísticas são executadas sobre essas instâncias até alcançar o valor-alvo ou terminar devido ao critério de parada original (número de gerações nos AEs e número de iterações nos GRASPs). Os resultados podem ser vistos nas Figuras 5.1 e 5.2.



(a)

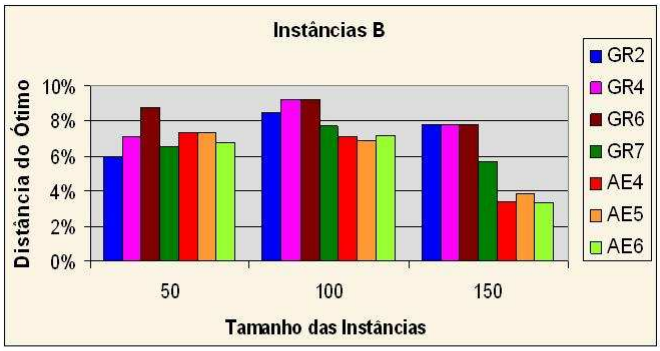


(b)

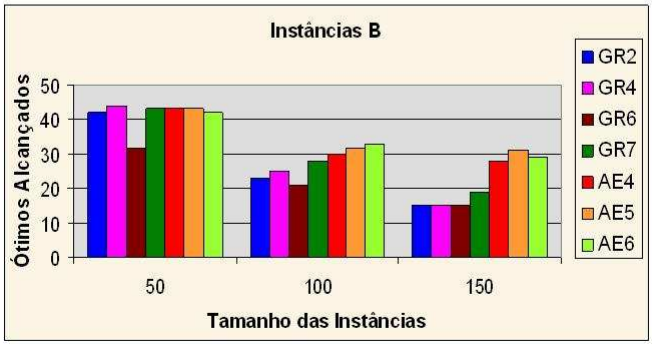
Figura 5.1: Resultados dos testes com instâncias pequenas do tipo A

Nas partes (b) das figuras, pode-se ver que a quantidade de ótimos alcançados diminui, logicamente, conforme aumenta a dificuldade das instâncias. Também pode ser visto que os AEs alcançaram, em geral, maior quantidade de ótimos que os GRASPs, o que de certa forma, pode ser ratificado com uma rápida análise na Tabela 5.7, onde, para as instâncias pequenas, os AEs conseguem ser tão bons ou melhores que os GRASPs.

Nas partes (a) das figuras, é mostrada a distância que a solução de cada meta-heurística ficou do ótimo (média das instâncias nas quais o ótimo não foi alcançado). As meta-heurísticas conseguiram ficar, em média, a 10% do ótimo, o que, para instâncias pequenas é um bom resultado, pois o valor do ótimo nestas instâncias é pequeno (variam



(a)



(b)

Figura 5.2: Resultados dos testes com instâncias pequenas do tipo B

de 20 a 500 unidades, em média). Esclarecendo: se o valor ótimo é 40, em uma destas instâncias, estar a 10% do ótimo significa obter um valor de 36.

As instâncias do tipo B, onde cada tarefa possui, em teoria, um maior número de predecessores, apresenta, conseqüentemente, um menor número de soluções possíveis em relação às instâncias do tipo A, pois o custo total de uma tarefa i em B é maior do que em A. Desta forma, é mais difícil conseguir recursos e ativar precedentes de forma que i possa ser ativada. Com um menor número de soluções possíveis, atingir o ótimo pode ficar um pouco menos difícil. Possivelmente, por isto, o desempenho das meta-heurísticas com as intâncias do tipo B foi um pouco superior às instâncias do tipo A (tanto na quantidade de ótimos alcançados quanto na distância média dos mesmos).

Um outro conjunto de testes mostra de maneira mais clara como é a distribuição das soluções geradas. Foram escolhidas 4 instâncias do tipo A de tamanhos 500, 700, 1200 e 2000. Em cada uma, as heurísticas propostas foram executadas e foi anotado o

valor da melhor solução encontrada por ela (não obrigatoriamente a solução ótima), num total de 1000 soluções geradas por cada uma (1000 iterações no GRASP e 50 gerações nos AEs, com 20 indivíduos em cada população). A partir disto, analisou-se a qualidade destas soluções em relação à melhor solução, para cada versão. Para facilitar a análise, as soluções foram divididas em faixas, de acordo com a diferença da mesma em relação à melhor solução encontrada. A Figura 5.3 mostra essas faixas (porcentagem em relação à melhor solução encontrada) no eixo X e o número de soluções que ficaram nessa faixa no eixo Y, nesta figura. Quanto mais concentrado perto de 0% melhor a capacidade do algoritmo de gerar sempre boas soluções.

A heurística GR6 apresentou o pior resultado entre as heurísticas testadas, gerando um bom número de soluções com mais 10% de distância em relação à melhor solução gerada por ela. Isto se deve pela maneira como a calibragem dos parâmetros (Calib4) é feita, fazendo com que soluções de qualidade ruim possam ser geradas sem nenhum tipo de controle.

Neste mesmo teste, a meta-heurística GR7 teve um bom resultado, gerando as soluções com no máximo 8% distância de sua melhor solução. A BL4, usada por esta versão do GRASP, é a uma das responsáveis por este desempenho, pois, através das suas várias tentativas de reconstrução da solução, dificulta que uma solução ruim seja apresentada pela meta-heurística. Também pode-se verificar isso, ao constatar que o GR4, cuja única diferença em relação ao GR7 é utilizar a BL3 ao invés da BL4, teve desempenho inferior, principalmente em (b).

Também apresentaram resultados interessantes as 3 versões de AEs. Isto se deve, possivelmente, a uma característica dos algoritmos de combinação que não consegue gerar filhos muito diferentes dos pais, ou seja, a qualidade das soluções do filho não conseguem ser muito diferente da qualidade dos pais, de maneira geral. Ou seja, a geração de soluções com qualidades "próximas" não quer dizer, necessariamente, que as soluções geradas são boas. No entanto, a característica de gerar soluções com pouca variação de qualidade é um fator importante ao se analisar uma (meta)heurística.

Outra forma interessante de verificar a diferença de resultados entre os GR4 e GR7, pode ser vista com este outro experimento: para cada uma das 500 iterações dos GRASPs, anotou-se quanto tempo foi gasto em cada etapa (heurística construtiva e buscas locais)

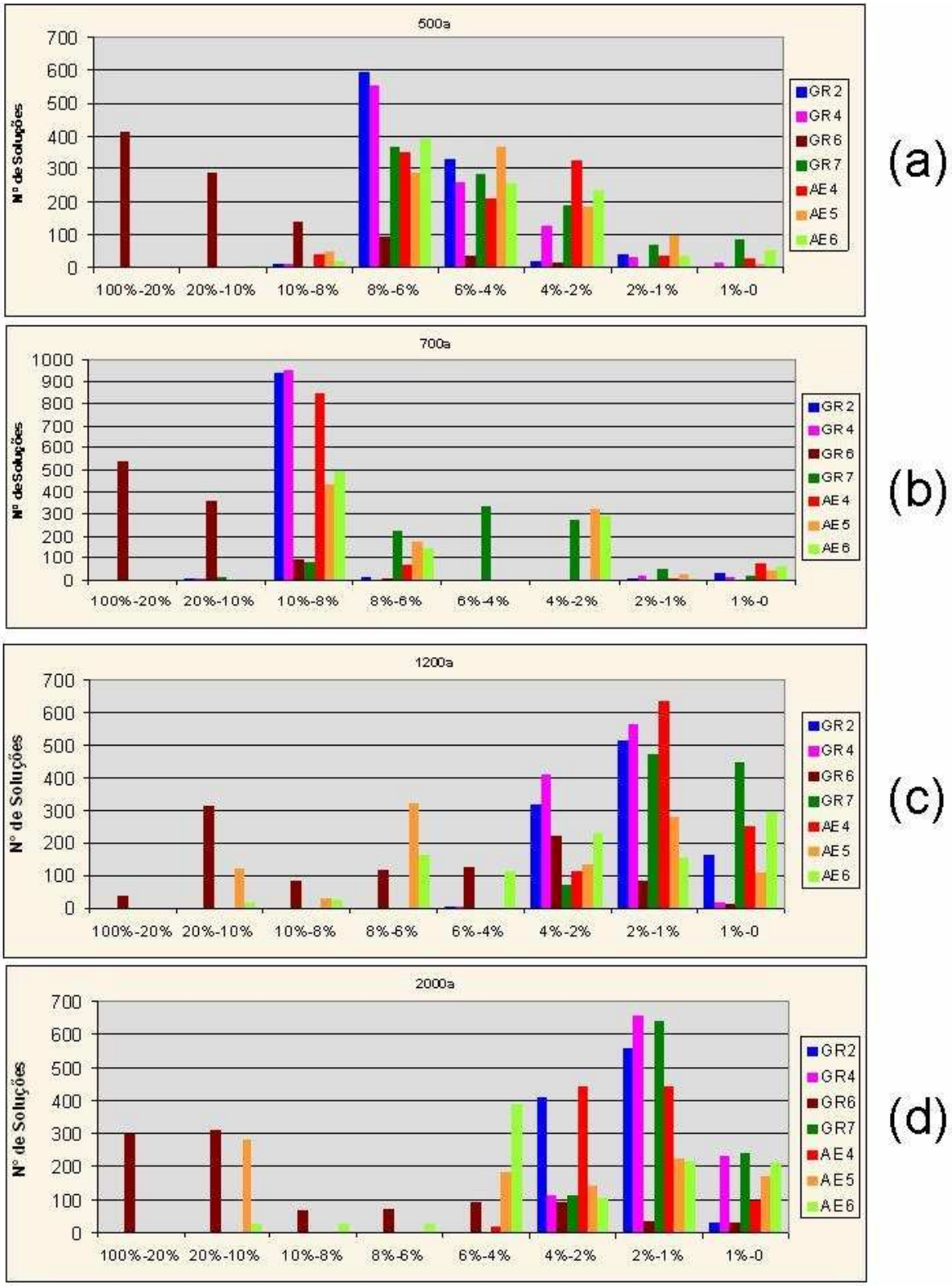


Figura 5.3: Faixas de qualidade das soluções das meta-heurísticas em relação à melhor solução encontrada

e quanto cada uma destas etapas contribui para a qualidade final da solução. Para este experimento foi utilizada a instância 700a.

A Figura 5.4 mostra o tempo gasto (a) e a contribuição de cada etapa (b). Da mesma forma, as figuras 5.5 e 5.6 mostram o experimento realizado com as heurísticas GR7 e GR6, respectivamente.

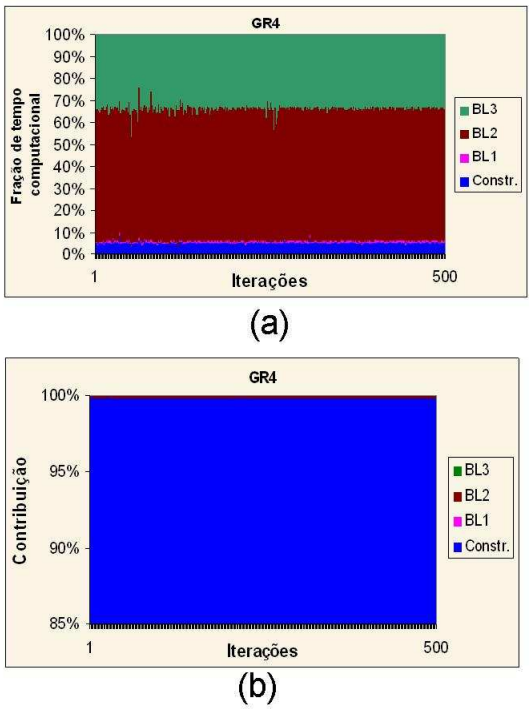


Figura 5.4: Porção de tempo gasto (a) e contribuições (b) no GR4

A BL4, embora consuma mais da metade do tempo gasto em cada iteração, contribui com aproximadamente 10% da solução final, no caso do GR7. O que definitivamente não ocorre com a BL3 que consome muito tempo (mais de 30%) e praticamente não melhora a qualidade da solução. Comportamento semelhante ocorre com a BL1, no entanto, esta não consome muito tempo. Portanto, não custa manter a BL1, pois como na Figura 5.6, em alguns casos ela consegue melhorar um pouco a solução.

A boa utilização do tempo passa a ser uma importante questão também. Em uma nova bateria de testes, será verificada como fica a convergência empírica das heurísticas, imposto um limite de tempo e também um valor-alvo. Novamente, foram utilizadas as instâncias 500a, 700a, 1200a e 2000a. O valor-alvo será uma média (aproximada) dos três melhores resultados obtidos pelas heurísticas para cada uma destas instâncias. Serão feitas 20 execuções em cada instância para cada algoritmo. A Figura 5.7 mostra os resultados obtidos. No eixo X estão cada uma das 20 execuções e no eixo Y o tempo gasto por cada

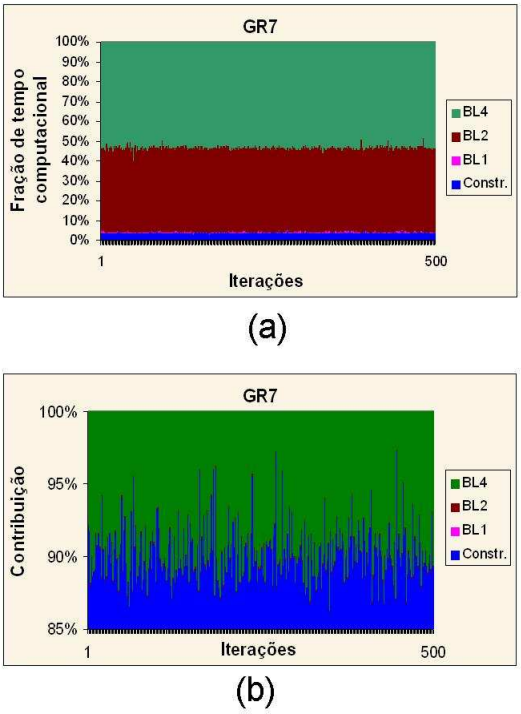


Figura 5.5: Porção de tempo gasto (a) e contribuições (b) no GR7

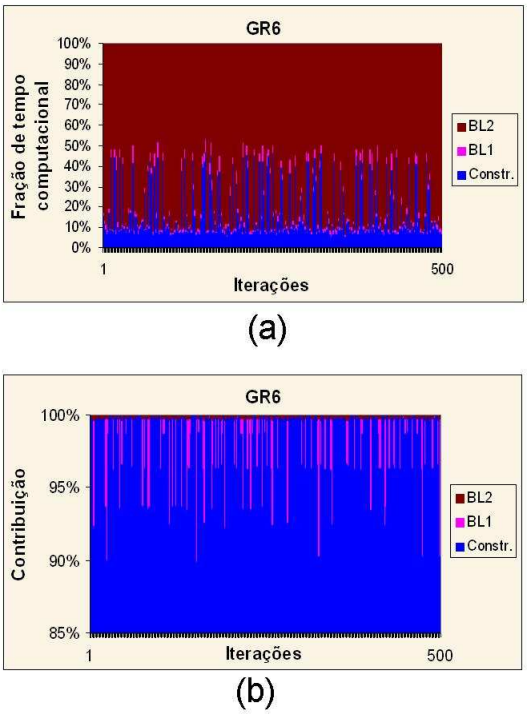


Figura 5.6: Porção de tempo gasto (a) e contribuições (b) no GR6

heurística para atingir o alvo. Quando o marcador incidir sobre o maior valor do eixo Y, isto indicará que a heurística não atingiu o alvo dentro do limite de tempo.

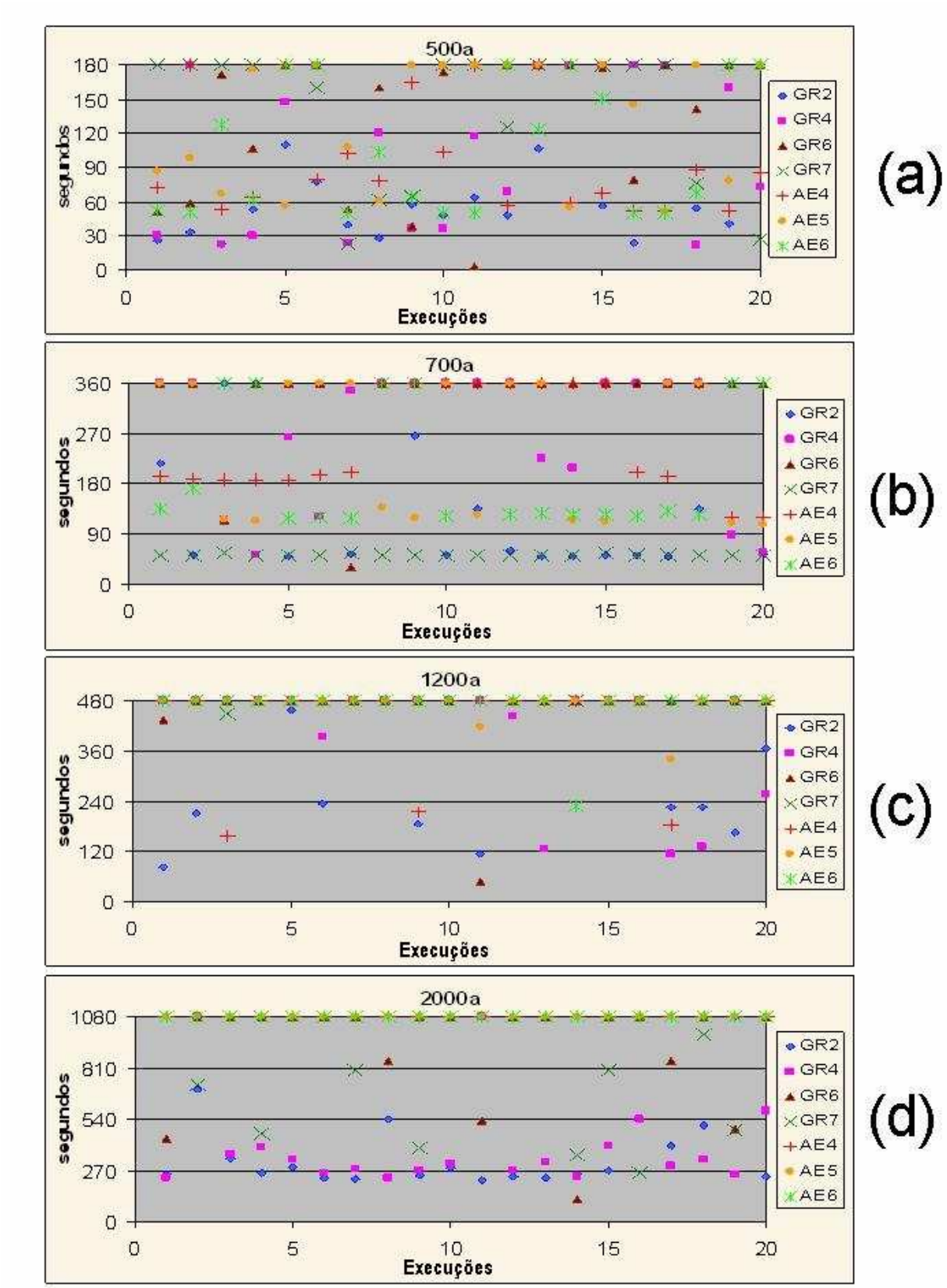


Figura 5.7: Convergência empírica dos AEs e GRASPs

Nas instâncias pequenas (quadros (a) e (b)), verificou-se um boa convergência empírica por parte das heurísticas AEs e GRASPs. Todas conseguiram chegar no alvo várias vezes, reafirmando que, para esse tipo de instâncias, tanto AEs como GRASPs têm bom desempenho. No entanto, para instâncias maiores (quadros (c) e (d)), isto já não se verifica. Na instância 1200a, houve apenas 6 convergências por parte dos AEs contra 18 dos GRASPs. Na instância 2000a, pode ser visto cenário ainda menos favorável para os AEs, já que estes não convergiram em nenhuma oportunidade. Este resultado, possivelmente, se deve ao fato de que os algoritmos de combinação, além de melhorarem pouco a qualidade das soluções filhos (em relação aos pais), consomem muito tempo computacional, especialmente quando precisam verificar quais tarefas estão disponíveis em cada solução pai (complexidade $O(n^2)$), o que é feito a cada iteração do algoritmo de combinação utilizado (Comb2). De maneira geral, a geração de uma nova população consome bastante tempo computacional e dificilmente consegue aprimorar a qualidade dos filhos de maneira satisfatória (e, principalmente, proporcional ao tempo gasto).

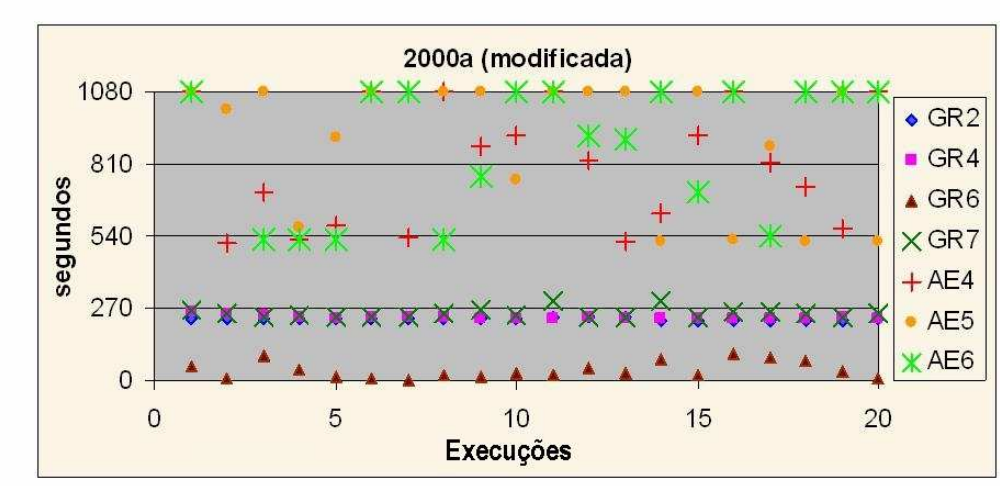


Figura 5.8: Convergência empírica dos AEs e GRASPs com alvo reduzido

Porém, isto não significa que os AEs tiveram desempenho tão ruim. Na Figura 5.8, a mesma instância 2000a foi repetida, mas o valor-alvo foi reduzido em 1% (na prática, 56 unidades). Sem exceções, as versões GRASPs alcançaram o alvo em todas as 20 execuções em menos de 270 segundos. Já os AEs começaram a alcançar o alvo por várias vezes. Ou seja, no teste original, os AEs não alcançaram o alvo, mas ficaram muito próximos dele (já que houve convergência com a redução de apenas 1%). Este "limite virtual", de chegar perto do alvo estabelecido, também ocorreu, de forma mais ou menos substancial, nas

instâncias menores. O que mais uma vez cria indícios para se presumir que a capacidade de melhoramento da população, por meio dos algoritmos de combinação propostos, tem um certo limite.

No último experimento realizado, novamente foi fixado um limite de tempo para cada uma das quatro instâncias testadas. O objetivo é verificar como vai evoluindo a qualidade da melhor solução encontrada pela heurística em questão, com o passar do tempo. Nas Figuras 5.9 a 5.12 são apresentados os gráficos gerados por este experimento.

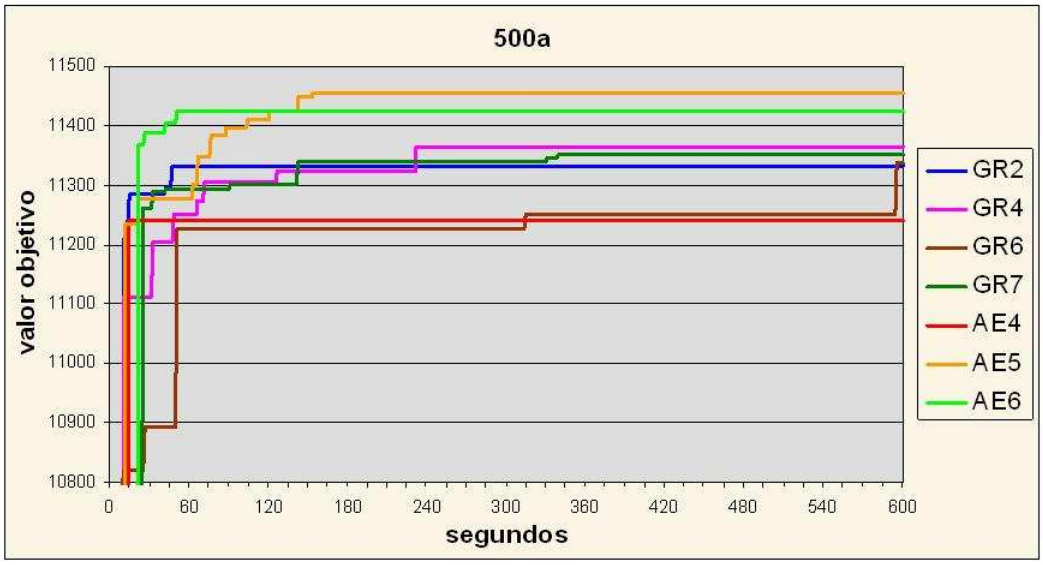


Figura 5.9: Evolução das soluções na instância 500a

Algumas observações podem ser feitas sobre estes gráficos. A primeira solução marcada pelos AEs acontece sempre um pouco depois das versões GRASPs porque a primeira marcação de tempo ocorre após a geração de toda a população inicial. Isto mostra que boa parte do tempo que é gasto pelos AEs é consumido nesta primeira geração.

De maneira geral, os AEs também só conseguem melhorar a qualidade da solução até metade do limite de tempo (300 segundos). Nas figuras 5.10 e 5.12, observam-se no máximo 3 melhorias. Nas figuras 5.9 e 5.11, observam-se mais evoluções, porém foram poucas após os 300 segundos.

Quanto à versões GRASP, estas melhorias ocorreram também após os 300 segundos (embora que, logicamente, em menor número do que antes deste tempo). Na instância

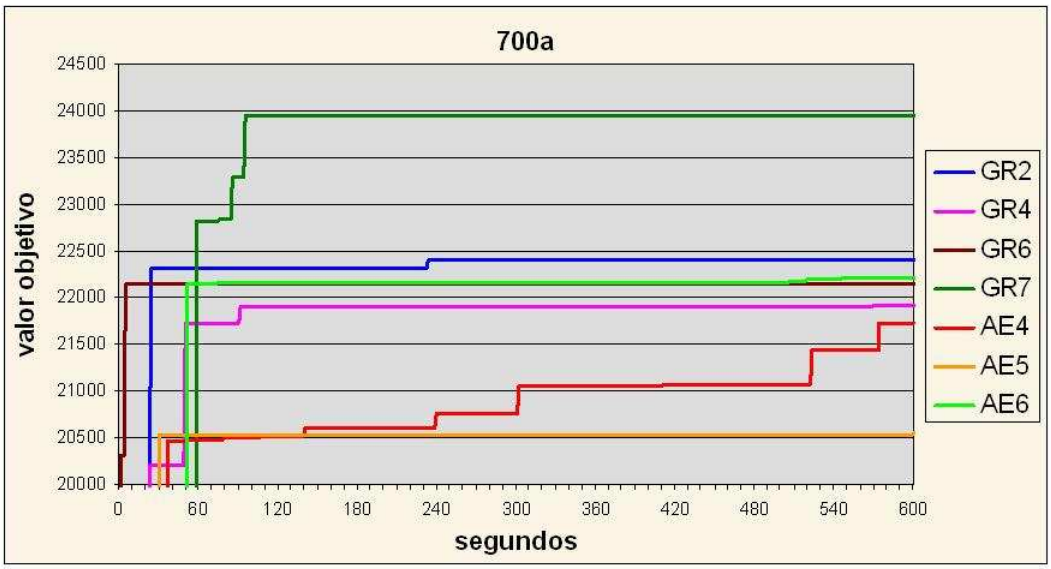


Figura 5.10: Evolução das soluções na instância 700a

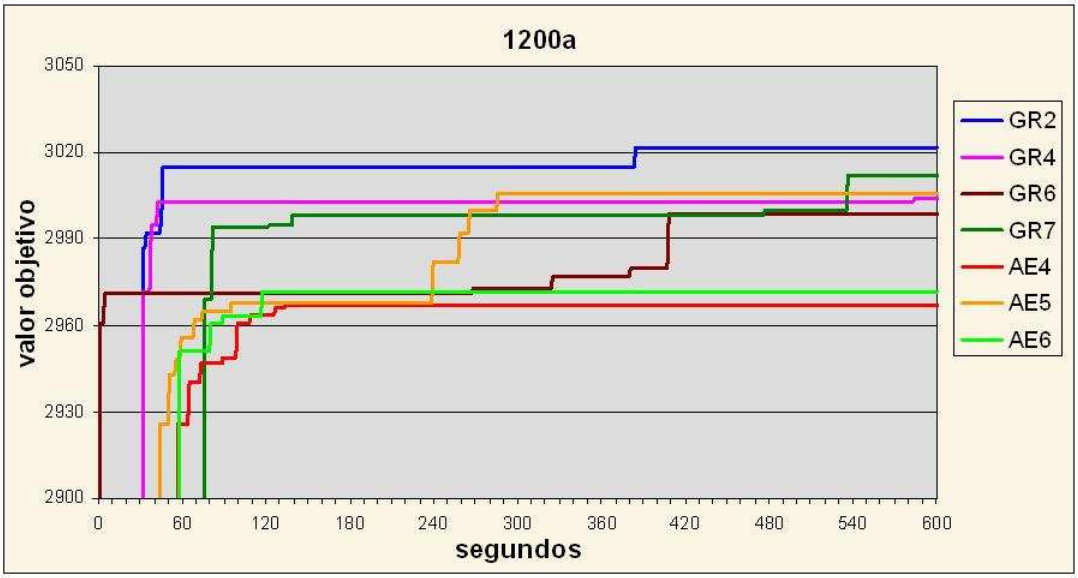


Figura 5.11: Evolução das soluções na instância 1200a

500a, mais uma vez, os AEs foram melhor que os GRASPs o que ocorreu de forma inversa na instância 2000a.

Uma outra análise também pode ser feita em relação ao *gap* de cada evolução. As versões GRASP conseguiam *gaps*, também de maneira geral, iguais ou maiores que os dos

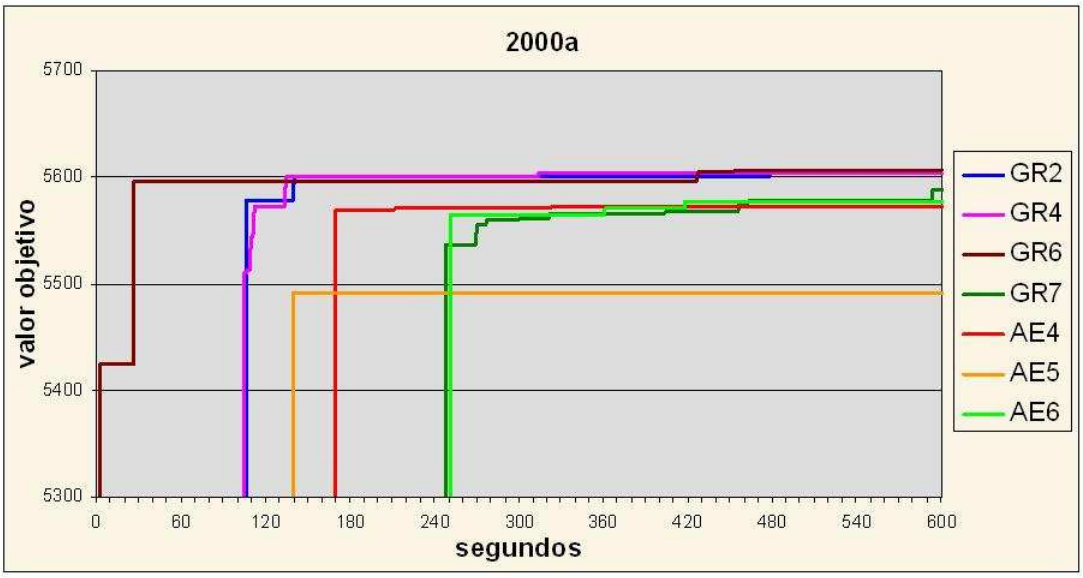


Figura 5.12: Evolução das soluções na instância 2000a

AEs. Isto se deve ao fato de que cada iteração no GRASP é independente das anteriores, enquanto nos AGs uma nova geração é feita (quase) sempre em função da anterior.

5.3 Instâncias E

Como já foi dito, estas instâncias têm a característica muito peculiar de não possuir arcos (precedência). Isto implica no fato de que resolvê-las se torna um pouco mais fácil, pois todas as restrições relacionadas com a precedência entre tarefas não existem mais. Computacionalmente, através de *softwares* como o GLPK, pode-se obter em tempo aceitável (alguns minutos) o resultado exato ou, pelo menos, limites inferior e superior bem próximos do valor ótimo. Por estas razões, os resultados destas instâncias serão apresentados separadamente.

Uma outra observação que deve ser feita é a de que, como não existem arcos, a calibração Calib2 calculará o α como sendo 0 (zero). Assim, a heurística construtiva ADDR passará a se comportar como a versão determinística ADD. Além do mais, não haverá sentido em se utilizar a fração de corte com valor diferente de 0 (zero). A utilização da fração de corte permite que uma tarefa que não retorne lucro líquido positivo faça parte da solução, visando disponibilizar suas sucessoras, possivelmente mais lucrativas. Como

não existem mais sucessoras, não há porque permitir que estas tarefas façam parte da solução. Da mesma forma, a utilização da folga não tem sentido.

A BL1 também perde o sentido, pois ela tenta remover tarefas que tenham sido aceitas visando suas sucessoras. A BL2 torna-se também sem sentido, pois ela tenta eliminar o conjunto de tarefas sucessoras de uma tarefa qualquer, conjunto este que para as instâncias E não existe. A BL3, como já foi dito, só terá sentido após a aplicação das outras buscas locais. Logo a única busca local que poderia ser utilizada nessas instâncias é a BL4. No entanto, o algoritmo de recriação utilizado pela BL4, não conseguiu melhorar os resultados. Assim, somente o GR6 pode ser usado sem grandes modificações na sua estrutura. Porém foi preciso fixar a fração de corte em 0 (zero) e a folga em 1 (um), como já foi explicado. As buscas locais BL1, BL2 e BL3 também foram retiradas dos algoritmos. A Tabela 5.10 mostra os resultados médios das heurísticas testadas com as intâncias E. As colunas LI e LS mostram, respectivamente, o limite inferior e o superior do valor ótimo obtido pelo GLPK. A última coluna mostra se um desses limites foi alcançado em alguma das execuções dos algoritmos testados. A Tabela 5.11 mostra o tempo computacional médio gasto (em segundos) pelas heurísticas em 3 execuções.

Pelos resultados mostrados nas tabelas, pode-se concluir que os algoritmos de combinação utilizados pelos AEs conseguem ser também pouco eficazes, principalmente, conforme vai aumentando o tamanho das instâncias testadas, fazendo com que os AEs fiquem em média a 3% do GR6 (nas três instâncias maiores). Além disso, o tempo de processamento dos AEs ficou mais de quatro vezes o tempo do GR6 também nestas últimas intâncias, mostrando que vale mais a pena reconstruir novas soluções ao invés de tentar combiná-las, neste tipo de instância.

n	GR6	AE4	AE5	AE6	LI	LS	Alcançado
50e	**176**	**176**	**176**	**176**	176	176	LS
100e	**556**	0,3%	0,3%	0,3%	557	557	LS
150e	**789**	**789**	**789**	**789**	789	789	LS
200e	0,2%	**993**	0,1%	0,1%	993	993	LS
250e	0,0%	0,2%	**1002**	0,2%	1003	1003	LS
300e	**2641**	0,4%	0,3%	0,5%	2641	2641	LS
350e	**2519**	0,3%	0,4%	0,5%	2521	2523	LS
400e	**3227**	0,5%	0,6%	0,6%	3230	3230	-
450e	**4581**	0,6%	0,6%	0,6%	4581	4584	LI
500e	**5415**	0,8%	0,7%	0,8%	5417	5420	-
550e	**5333**	0,8%	1,0%	1,0%	5334	5337	LI
600e	**6013**	0,7%	0,8%	0,7%	6018	6018	-
650e	**8301**	1,0%	0,9%	1,2%	8306	8309	-
700e	**6981**	1,0%	1,4%	1,2%	6986	6989	-
750e	**6506**	1,2%	0,8%	1,3%	6506	6508	LI
800e	**8385**	1,3%	1,4%	1,3%	8388	8392	LI
850e	**8707**	1,6%	1,4%	1,8%	8712	8716	-
900e	**10951**	1,2%	1,3%	1,3%	10948	10958	LI
950e	**13343**	1,4%	1,8%	1,3%	13342	13349	LI
1000e	**13634**	1,3%	1,4%	1,5%	13635	13641	LI
1100e	**15477**	1,3%	1,6%	1,5%	15477	15480	LI
1200e	**21127**	1,8%	1,6%	1,8%	21115	21134	LI
1300e	**23524**	2,5%	2,4%	2,1%	23506	23533	LI
1400e	**23188**	1,9%	1,8%	2,1%	23175	23207	LI
1500e	**25782**	2,7%	2,7%	2,6%	25757	25789	LI
1600e	**30784**	2,3%	2,1%	2,2%	30758	30792	LI
1700e	**29572**	2,4%	2,4%	2,3%	29555	29576	LI
1800e	**38640**	3,2%	3,0%	3,0%	38631	38652	LI
1900e	**36726**	2,9%	3,3%	3,2%	36713	36733	LI
2000e	**43138**	3,2%	2,8%	2,7%	43127	43143	LI

Tabela 5.10: Resultados médios para as instâncias E

n	GR6	AE4	AE5	AE6
50e	0,3	0,7	1,0	0,8
100e	1,0	2,0	2,1	2,0
150e	1,3	3,3	3,5	3,6
200e	2,5	6,3	6,1	9,0
250e	3,9	9,5	9,0	9,5
300e	6,3	14,2	14,4	22,9
350e	8,1	18,8	19,0	35,1
400e	10,6	23,6	24,0	43,7
450e	15,8	33,2	32,9	59,4
500e	19,4	42,6	42,1	73,8
550e	22,8	48,5	48,7	84,7
600e	26,7	56,7	57,7	101,1
650e	35,9	75,8	76,9	133,1
700e	41,4	89,5	90,7	152,9
750e	46,6	102,5	101,4	175,0
800e	53,1	114,6	116,8	191,4
850e	60,3	131,4	130,6	213,9
900e	75,7	157,5	151,3	267,4
950e	83,2	175,4	181,1	279,1
1000e	95,4	196,8	196,3	364,3
1100e	102,1	226,8	263,7	381,5
1200e	119,1	307,9	299,2	493,0
1300e	137,1	363,3	462,3	552,5
1400e	150,9	401,7	656,6	688,3
1500e	202,6	470,1	768,7	579,2
1600e	227,0	574,1	921,3	560,2
1700e	241,6	705,3	854,5	654,6
1800e	270,6	1279,5	677,3	695,1
1900e	309,1	1474,2	829,9	839,4
2000e	363,8	1641,6	896,0	932,7

Tabela 5.11: Tempos computacionais médios (em segundos) para as instâncias E

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho, foi apresentada uma nova modelagem para o PETRR, aqui denominado de PETRRD (Problema de Escalonamento da Tarefas com Restrição de Recursos Dinâmicos). Embora a motivação para esta modelagem tenha vindo de um jogo de computador, esta pode ser aplicada a grandes projetos de expansão de empresas, desde que ele possa ser feito em etapas e que seja admitido o retorno de algum tipo de lucro antes do fim do projeto.

Por se tratar de uma modelagem nova e, por conseqüência, terem sido utilizadas instâncias especialmente geradas para este trabalho, não foi possível comparar os algoritmos aqui propostos com outros da literatura. No entanto, algumas conclusões podem ser feitas a respeito dos mesmos com base nos resultados obtidos.

Uma boa calibração dos parâmetros adotados na heurística randomizada ADDR, pode levar a melhorias significativas nos resultados. O parâmetro α em função da densidade de arcos do grafo de entrada (Calib2) apresentou melhores resultados do que as demais calibrações propostas, por não estar fixo em alguns valores (Calib1) e por não ser tão difícil ajustar um fator de correção para ele.

A fração de corte também desempenha papel significativo, principalmente, ao se tratar das instâncias do tipo E (sem arcos). A utilização de um valor bem específico (zero) consegue melhorar bastante a qualidade das soluções. De maneira geral, as chamadas técnicas auxiliares se mostraram importantíssimas para a obtenção de resultados melhores com gasto de tempo computacional bem pequeno.

As buscas locais apresentaram resultados bem distintos. A BL1 e BL3 praticamente

não melhoram a qualidade das soluções, em algumas versões do GRASP; enquanto uma consome pouco tempo computacional a outra, no entanto, desperdiça até 35% do tempo gasto. A BL4, uma das que mais consome tempo, consegue apresentar melhores resultados quando a instância em questão permite elevado número de tarefas ativadas.

Os AEs, que nos testes com instâncias pequenas obtiveram bons resultados, perdem bastante de sua eficiência conforme aumenta o número de tarefas a serem tratadas. Isto se deve, em parte, pela dificuldade apresentada pelos mesmos em conseguir contínuas atualizações da melhor solução. Para tentar diminuir este problema foram aplicadas técnicas de perturbação e escolha mais elaboradas para os pais. Outro problema apresentado pelos AEs é o grande consumo de tempo, principalmente para a geração da população inicial.

As versões GRASPs obtiveram os melhores resultados em instâncias maiores porque, de forma diferente dos AEs, suas iterações são completamente independentes e, portanto, a geração de uma solução ruim numa iteração não interfere na geração de outras soluções posteriormente. Isto permite que o GRASP consiga atualizar a melhor solução um maior número de vezes. Essas melhorias conseguem, em muitos casos, ser ainda maior nas versões GRASPs do que nos AEs.

Em relação às instâncias do tipo E, pôde-se perceber que parte dos algoritmos propostos possui parâmetros considerados ótimos, como já foi dito. Além disso, as buscas locais BL1, BL2 e BL3 são inócuas com esta classe de instâncias. Dentre as 7 heurísticas testadas mais amplamente, o GR2 perderia sentido, pois este faria diversas iterações idênticas, e o GR7, que teria como trunfo a reconstrução das soluções, apresentou resultados ruins. Os AEs 4,5 e 6 testados também não conseguiram ser competitivos com o GR6 nem na questão da qualidade das soluções menos ainda quanto ao tempo computacional exigido. Esta diferença se mostra ainda mais evidente com o aumento do tamanho das instâncias.

De maneira geral, na classes de instâncias testadas, as versões GRASP conseguiram melhor desempenho (tanto no tempo quanto na qualidade das soluções) que os AEs, pois os algoritmos de combinação consomem um tempo desproporcional em relação às melhorias que conseguem para a qualidade final da solução. Em resumo, a utilização das técnicas auxiliares, a boa calibração dos parâmetros e a constante "construção e melhoria da solução" se mostraram a melhor estratégia heurística a ser utilizada para tratar as instâncias do PETRRD.

6.1 Trabalhos Futuros

Como possíveis trabalhos futuros neste tema podem ser incluídos:

- Teste com novos tipos de instâncias: árvores binárias (*in-tree* e *out-tree*), grafos diamante, *clusters* de tarefas, entre outras.
- Utilização de outros objetivos como: menor tempo de ativação de todas as tarefas (*makespan*), maximização do lucro (ao invés da quantidade de recursos disponíveis), entre outros.
- Testes com novas estratégias de calibração baseadas em informações provenientes do grafo de entrada.
- Utilização de critérios de inserção das tarefas na LRC baseando-se também em informações sobre seus sucessores.
- Utilização de outras meta-heurísticas, como Busca Tabu ou VNS (*Variable Neighborhood Search*).
- Algoritmos paralelos para obtenção de resultados em tempo computacional menor.

Referências

[1] REMY, J. Resource constrained scheduling on multiple machines. *Information Processing Letters*, v. 91, p. 177–182, 2004.

[2] KALINOWSKI, T.; KORT, I.; TRYSTAM, D. List scheduling of general task graphs under LogP. *Parallel Computing*, v. 26, p. 1109–1128, 2000.

[3] BOULEIMEN, K.; LECOCQ, H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, v. 149, p. 268–281, 2003.

[4] GONÇALVEZ, J. F.; MENDES, J. J. M.; RESENDE, M. G. C. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research(submitted)*, 2005.

[5] RABBANI, M. et al. A new heuristic for resource-constrained project scheduling in stochastic networks using critical chain concept. *European Journal of Operational Research(accepted)*, 2005.

[6] BRUCKER, P. et al. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, v. 12, p. 3–41, 1999.

[7] GONÇALVES, J.; MENDES, J.; RESENDE, M. A random key based genetic algorithm for the resource constrained project scheduling problems. *International Journal of Production Research (submitted)*, 2005.

[8] ARTIGUES, C.; MICHELON, P.; REUSSER, S. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, v. 149, p. 249–267, 2003.

[9] MIKA, M.; WALIGÓRA, G.; WEZGLARZ, J. Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, v. 164, p. 639–668, 2005.

[10] SENKUL, P.; TOROSLU, I. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, v. 30, p. 399–422, 2005.

[11] ABEYASINGHE, M. C. L.; GREENWOOD, D. J.; JOHANSEN, D. E. An efficient method for scheduling construction projects with resource constraints. *International Journal of Project Management*, v. 19, p. 29–45, 2001.

[12] FOULDS, L. R.; WILSON, J. M. Scheduling operations for the harvesting of renewable resources. *Journal of Food Engineering*, v. 70, p. 281–292, 2005.

[13] ZHANG, H. et al. Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction*, v. 14, p. 393–404, 2005.

[14] ABDEDDAÏM, Y.; ASARIN, E.; MALER, O. Scheduling with timed automata. *Theoretical Computer Science (accepted)*, 2004.

[15] MARTELLO, S.; TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, 1990.

[16] KELLER, C. P. Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, v. 41, p. 224–231, 1989.

[17] LAPORTE, G.; MARTELLO, S. The selective travelling salesman problem. *Discrete Applied Mathematics*, v. 26, p. 193–207, 1990.

[18] BEASLEY, J. E. Population heuristics. In: PARDALOS, P.; RESENDE, M. (Ed.). *Handbook of Applied Optimization*. Oxford: Oxford University Press, 2002. p. 138–157.

[19] FESTA, P.; RESENDE, M. GRASP: An annotated bibliography. In: RIBEIRO, C.; HANSEN, P. (Ed.). *Essays and Surveys on Metaheuristics*. [S.l.]: Kluwer Academic Publishers, 2002. p. 325–367.

[20] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: [s.n.], 1975.

[21] GONÇALVES, J.; MENDES, J.; RESENDE, M. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, v. 167, p. 77–95, 2005.

[22] BEASLEY, D. *FAQ (A guide to frequently asked questions)*. Acessado em Janeiro/2006. Disponível em: <http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/top.html>.

[23] DAVIS, L. (Ed.). *Handbook of Genetic Algorithms*. New York, NY: Van Nostrand Reinhold, 1991. (VNR Computer Library).

[24] REEVES, C. R. (Ed.). *Modern Heuristic Techinques for Combinatorial Problems*. Osney Mead, Oxford: Blackwell Scientific Publications, 1993. (Advanced Topics in Computer Science).

[25] MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. New York, NY: Springer-Verlag, 1996.

[26] FEO, T.; RESENDE, M. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[27] AIEX, R. et al. GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, v. 17, n. 2, p. 224–247, 2005.

[28] RESENDE, M.; RIBEIRO, C. Grasp with path-relinking: Recent advances and applications. In: IBARAKI, T.; NONOBE, K.; YAGIURA, M. (Ed.). *Metaheuristics: Progress as Real Problem Solvers*. [S.l.]: Springer, 2005. p. 29–63.

[29] RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). *in Handbook of Metaheuristics*. [S.l.]: Kluwer Academic Publishers, 2003. p. 219–249.

[30] GLPK home page. Acessado em Janeiro/2006. Disponível em: <http://www.gnu.org/software/glpk/glpk.html>.