Proposta e Análise Experimental de Heurísticas GRASP para um problema de escalonamento de tarefas com recursos dinâmicos

André Renato Villela da Silva
Instituto de Computação - Universidade Federal Fluminense
Niterói - RJ
avillela@ic.uff.br

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

RESUMO

Neste artigo é proposto o Problema de Escalonamento de Tarefas com Restrições de Recursos Dinâmicos (PETRRD) como um novo modelo para o Problema de Escalonamento de Tarefas com Restrições de Recursos (PETRR). No modelo proposto, nós coletamos recursos (lucros) de acordo com o momento em que cada tarefa é ativada durante o período de planejamento. Para resolver o PETRRD, nós propusemos uma formulação matemática, Técnicas Aprimorantes e heurísticas GRASP.

PALAVRAS CHAVE: metaheurística, problema de escalonamento de tarefas, GRASP. Otimização Combinatória.

ABSTRACT

In this paper is proposed the Dynamic Resource-Constrained Task Scheduling Problem (DRCTSP) as a new model for the Resource Constrained Task Scheduling Problem (RCTSP). In the proposed model, we gather resources (profits) accordingly with the moment in which each task is activated during the planning process. To solve the DRCTSP, we proposed a mathematical formulation, enhancement techniques and GRASP heuristics.

KEYWORDS: metaheuristic, task scheduling problem, GRASP. Combinatorial Optimization.

1- Introdução

O problema de escalonamento de tarefas (PET) é muito conhecido e muito estudado já há bastante tempo. Consiste basicamente em um conjunto T de tarefas que devem ser executadas em um conjunto P de unidades de processamento o mais rápido possível. O modelo clássico pressupõe também precedências entre algumas tarefas, ou seja, uma tarefa *i* só pode ser executada depois que todas as suas tarefas predecessoras P(i) já tenham sido. Caso uma tarefa *i* seja executada numa unidade *m* e uma tarefa *j*, sucessora de *i*, seja executada em *p*, pode ser preciso enviar informações de *m* para *p*. Para que isto aconteça, os dois processadores vão precisar reservar um tempo de processamento para enviar e receber a mensagem. Durante este tempo, no modelo clássico, eles não poderão processar as tarefas. Modelos mais gerais trabalham também com unidades de processamento distintas (ambiente heterogêneo) em contraposição ao modelo onde todas as unidades de processamento são idênticas (ambiente homogêneo).

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 (com demandas distintas, inclusive) (veja Brucker(1999) e Gonçalves(2005)). 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), como descrito em Brucker(1999).

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 aplicativos 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 (Artigues(2003)). 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 encontradas em várias sub-áreas da Otimização Combinatória, como podem ser vistas em: Brucker(1999), Gonçalves(2005), Mika(2005), Senkul(2005), Abeyasinghe(2001) e Foulds(2005).

Neste trabalho é apresentada uma nova modelagem para o problema de escalonamento de tarefas, aqui denominada *PETRRD* (Problema de Escalonamento de Tarefas com Restrição de Recursos Dinâmicos). Para sua solução, são propostas, além do modelo matemático para obter soluções exatas para pequenas instâncias, duas versões da heurística GRASP com o objetivo de gerar bons limites inferiores de um valor ótimo. Também são apresentados mecanismos chamados de *Técnicas Aprimorantes* com o objetivo de melhorar o desempenho dos algoritmos aqui propostos.

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: na Seção 2, são explicadas as características do modelo proposto e a formulação matemática; na Seção 3, são mostradas as técnicas Aprimorantes e as heurísticas propostas; as instâncias que foram geradas para este problema e os resultados e análise computacional são mostradas na Seção 4; na Seção 5, por fim, são apresentadas algumas conclusões obtidas a partir dos experimentos realizados.

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). A motivação para elaboração desta modelagem veio de um jogo de computador chamado *Civilization III*, da empresa *FIRAXIS Games INC*. 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 técnicas de inteligência artificial para determinar as ações a serem tomadas.

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.

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 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 vista a seguir.

2.1- O modelo computacional

Suponha um grafo direcionado e acíclico $G = (V,A)$, 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 H unidades de tempo (ex: dias, semanas, meses). O objetivo é escolher quais tarefas escalonar e quando fazê-lo para que ao final das H unidades de tempo, haja a maior quantidade de recursos disponíveis. Ou seja, neste modelo nem todas as tarefas devem necessariamente ser escalonadas (devido à limitação de recursos disponíveis a cada unidade de tempo).

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 desta modelagem proposto em relação aos existentes na literatura. Em Bouleimen(2003), 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 possui 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.

2.2- Conceitos Básicos

Ativação: é a incorporação de uma tarefa i na solução do problema. Para tanto, se paga 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 H (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.
Custo (simples) c_i: é a quantidade de recursos necessária para a ativação de uma tarefa i.
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 (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.

2.3- Formulação matemática do PETRRD

Neste trabalho também é proposta uma formulação matemática do PETRRD, descrevendo-o como um Problema de Programação Inteira Mista. 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_0 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 na Figura 1.

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 (H). As equações (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), a partir de $t-1$: é a soma dos recursos inicialmente disponíveis em $t-1$, subtraído dos custos para a ativação das tarefas no intervalo atual e somado ao lucro acumulado até então (L_{t-1}). Vale observar que o valor inicial dos recursos disponíveis Q_0 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.

Max:

$$Q_H + L_H \tag{2.1}$$

S.T.

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

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

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

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

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

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

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

$$Q_t, L_t \in \mathbb{N} \quad \forall t = 0, ..., H \tag{2.9}$$

Figura 1: Formulação matemática para o PETRRD

Para ilustrar alguns destes conceitos, será mostrada a Figura 2, a seguir. Em branco, estão as tarefas já 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 $Q_0 = 2$. O horizonte de tempo é $H = 3$, ou seja, até a parte (c) da figura.

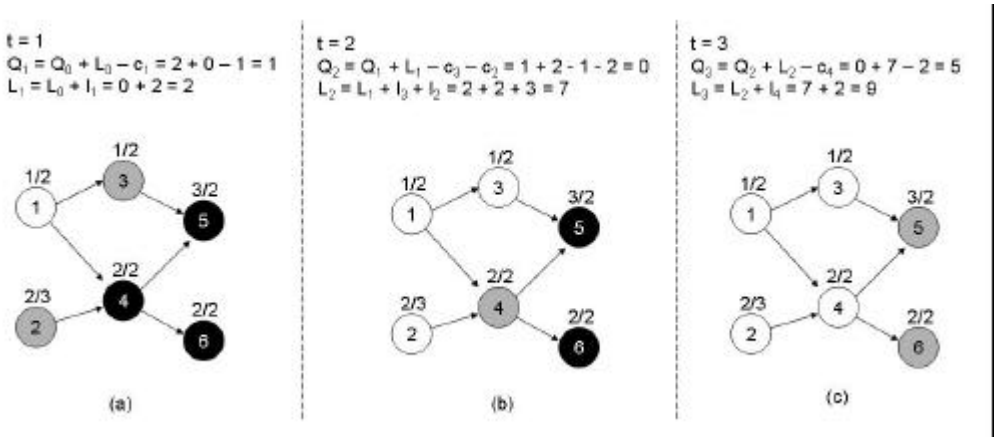

Figura 2: Exemplo de construção de uma solução

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 cin-

za). 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 de lucro). Os recursos disponíveis, no entanto, são obtidos pelos recursos que vieram de $t = 1$ (1 unidade de recurso) 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 $t = H$, encerra-se o processamento. O valor desta solução encontrada é facilmente calculado: soma-se o que vem de Q_3 (5 unidades) com o lucro L_3 (9 unidades) totalizando 14 unidades.

3- Heurísticas para o PETRRD

Como o PETRRD é uma generalização do problema de escalonamento de tarefas com restrições de recursos, ele pertence à classe dos problemas NP-Completo. Desta forma, técnicas aproximadas ou heurísticas tendem a ser alternativas viáveis para a sua solução, já que obter o resultado ótimo pode ser computacionalmente inviável em instâncias de grande porte.

Em princípio, para resolver este problema, pode-se utilizar um critério de priorizar tarefas muito simples: seu *custo-benefício*. 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. Assim, criou-se a heurística randômica ADDR.

Esta heurística é composta por dois laços. O laço externo trabalha a cada instante de tempo do problema (H iterações), criando uma lista de tarefas disponíveis e ordenando-as (crescentemente) de acordo com seu custo-benefício. A partir desta lista, seleciona-se um subconjunto com as $p\%$ melhores tarefas (este $p\%$ funciona como uma Lista Restrita de Candidatos LRC, da fase construtiva do GRASP e será chamado de "alfa" - α). Dentre essas tarefas, uma será escolhida aleatoriamente para fazer parte da solução. Se a tarefa possuir custo menor ou igual à quantidade de recursos disponíveis neste instante, a tarefa será ativada e os valores de Q e L são atualizados, senão ela será descartada. O processamento das tarefas disponíveis (laço interno) continua até que não haja mais tarefas disponíveis ou até que não haja mais recursos disponíveis. Esta heurística gulosa e randômica será utilizada na fase de construção do GRASP.

3.1 Técnicas Aprimorantes

Algoritmos construtivos do tipo ADDR normalmente não retornam necessariamente soluções que representam ótimos locais. Para tentar melhorar o desempenho deste construtivo básico, tanto na melhora da sua convergência como na melhora da qualidade da solução gerada, serão utilizadas algumas técnicas adicionais que visam impedir ou possibilitar a escolha de determinadas tarefas, como visto a seguir.

Eliminação de Arcos: *Por meio desta técnica de redução, 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). Se isto puder ser feito, ou seja, se houver outro caminho que ligue i a j, a precedência explícita em (i,j) estará implícita por este caminho. Esta técnica visa reduzir o tempo de execução da ADDR, eliminando informações redundantes.*

Ponderação Prévia: *Antes do início da ADDR, estima-se qual o menor tempo possível de início de cada tarefa i de acordo com a topologia do grafo de entrada: as tarefas sem predecessores têm tempo de início igual a 1; para as demais tarefas, o tempo de início é calculado como sendo o maior tempo de início dos predecessores diretos acrescidos 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 de custo-benefício (c_i / l_i), mas sim o custo c_i dividido pelo lucro total estimado.

***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 razão para considerar os casos de igualdade, é que a ativação de uma tarefa mesmo não dando lucro positivo irá possibilitar que seus sucessores sejam ativados nos tempos seguintes. A fração de corte pode ser utilizada em qualquer instante de tempo t do horizonte de tempo em questão. Ou seja, a partir deste instante, este novo critério de aceitação da tarefa estará valendo.*

***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 multiplicado por um valor (a folga) seja maior ou igual ao seu custo.*

3.2 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 contêm 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.

3.2.1 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. Assim, começa-se o processamento com as tarefas ativadas por último. Para cada uma destas tarefas é calculado o lucro total trazido por ela e subtraído seu custo. Se o resultado (lucro líquido) for positivo ou zero esta tarefa é mantida, caso contrário removida. O procedimento continua com as tarefas ativadas num instante imediatamente anterior até que todas as tarefas tenham sido analisadas. Retirando-se, portanto, as tarefas que consumiram recursos sem retornarem benefícios.

3.2.2 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 conjunto, levando em conta o tempo de ativação das tarefas envolvidas. Isto é interessante porque, às vezes, uma tarefa pouco lucrativa é ativada visando uma sucessora mais lucrativa (escolha causada pela folga, por exemplo). No entanto, esta sucessora pode não conseguir compensar o alto custo da tarefa antecessora, embora

consiga compensar seu próprio custo. Por fim, estas duas tarefas podem estar, em conjunto, gastando mais recursos do que os lucros gerados e pode ser interessante removê-las.

3.2.3 Busca Local 3 (BL3)

A BL3 funciona de modo bem diferente às anteriores: ela escolhe aleatoriamente um tempo t_k do intervalo [*H/4*, *H/2*]. Então todas as tarefas que forem ativadas depois de t_k, são removidas da solução. Passa-se então à fase de reconstrução da solução, porém, não será mais utilizado o critério guloso (ordenação e escolha das tarefas, com preferência àquelas com melhor custo-benefício) e sim será dada uma probabilidade proporcional ao lucro de cada tarefa disponível para ativação. As tarefas disponíveis serão escolhidas de acordo com esta probabilidade e quando tiverem sido gastos os recursos ou quando não houver mais tarefas para ativar, passa-se para o próximo instante de tempo (de forma semelhante ao algoritmo ADDR). Este critério probabilístico é utilizado para se tentar evitar a recriação da solução inicial. São feitas 10 tentativas com a solução original (para cada uma delas executa-se a BL1 na solução recriada) e se retorna a melhor solução obtida.

3.3 GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*) proposta por Feo e Resende (Feo(1995)) é um método iterativo que 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 de forma iterativa), 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, 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, muito utilizada em várias versões de GRASP na literatura (veja, por exemplo, Aiex(2005)), como também formas híbridas que tem se mostrado mais eficientes que as versões tradicionais (veja Bastos(2005), Silva(2006), Souza(2003)).

Para analisar o impacto da utilização das *Técnicas Aprimorantes* no resultado final das soluções geradas, foram propostas duas versões de GRASP. No GRASP1 não serão utilizadas a Fração de Corte, a Folga e a Ponderação Prévia, que são utilizadas no GRASP2. A utilização da Eliminação de Arcos se mostrou eficaz principalmente quando o grafo de entrada é muito denso (possuindo muitos arcos) e, portanto, será utilizado pelas duas versões, uma vez que ele não influencia na qualidade da solução obtida, a princípio.

Ambos os GRASPs terão como critério de parada inicial o número de 300 iterações. O GRASP2 utilizará a BL1 em todas as soluções geradas e a BL2 somente quando o valor da solução (gerada depois de BL1) for no mínimo 90% da melhor solução encontrada até o momento.

Para escolher os valores de α e dos parâmetros das *Técnicas Aprimorantes* utilizadas será utilizada a seguinte estratégia: será definido um intervalo de possíveis valores para cada parâmetro; cada valor deste intervalo será testado com a ADDR 20 vezes e o valor que retornar o melhor resultado será escolhido. Os parâmetros serão testados na seguinte ordem e com os seguintes intervalos: α-*[0,05; 1,0]* em incrementos de *0,05*; Fração de Corte-*[0,2; 1,0]* em incrementos de *0,1*; Folga-*[1,0; 1,4]* em incrementos de *0,1*; Ponderação Prévia-*["sim", "não"]*. Estes testes serão realizados antes das iterações do GRASP, servindo como um pré-processamento para cada instância testada.

4- 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*.

Como o modelo é uma proposta deste trabalho, não existem instâncias na literatura que possam ser aplicadas, inclusive porque as tarefas destas instâncias existentes não possuem o valor de lucro associado. Desta forma fica sem sentido a adaptação das mesmas e, portanto, foram utilizados dois tipos de instâncias geradas especificamente para o PETRRD. Nas instâncias do tipo A, 10% das tarefas não possuem precedentes. As demais possuem entre 1 e 5 precedentes escolhidos aleatoriamente. Nas instâncias do tipo B, a primeira tarefa não tem precedente. A partir da segunda tarefa, cada uma terá probabilidade de 20% de ser sucessora das tarefas anteriores (cada tarefa anterior será testada separadamente).

Os custos e lucros das tarefas variam de 1 a 50 e de 1 a 10, respectivamente. Estes valores foram definidos baseando-se em testes preliminares e têm por objetivo proporcionar que haja recursos suficientes para que uma boa porcentagem das tarefas seja ativada. Se os custos forem menores, permite-se com certa facilidade que todas as tarefas sejam ativadas e não fará sentido escolher quais tarefas farão parte da solução, pois haverá recursos para que todas sejam ativadas. Embora, isto possa parecer interessante, na verdade simplifica muito o trabalho de qualquer método heurístico, nivelando todos eles por baixo. No entanto, se os custos forem maiores, ocorrerá um problema inverso: pouquíssimas tarefas poderão ser ativadas, pois será necessário gastar uma quantidade maior de recursos. De forma análoga, não haverá muitas opções a serem feitas, reduzindo também o trabalho de qualquer heurística. Ou seja, os valores, conforme foram definidos, proporcionam que haja escolhas a serem feitas em relação a quais tarefas serão ativadas, tendo-se a necessidade de se gerenciar bem os recursos disponíveis, pois poderá não ser possível ativar muitas tarefas. Desta forma, o problema (ou melhor, a instância) fica mais interessante e mais difícil de obter valores próximos do ótimo.

A quantidade de recursos iniciais é sempre maior do que a tarefa sem predecessor de menor custo, acrescido de até 10 unidades (todos os valores escolhidos aleatoriamente com probabilidade uniforme dentro dos intervalos). Nas instâncias com até 1000 tarefas, a quantidade de unidades do horizonte de tempo será a raiz quadrada do número de tarefas. Nas instâncias maiores, será utilizada a raiz cúbica. Estas funções foram escolhidas também de acordo com a porcentagem de tarefas que poderão ser ativadas. Quanto maior o horizonte de tempo, maior será o lucro gerado, maior a quantidade de recursos disponíveis e maior o número de tarefas que poderão ser ativadas. Quanto menor o horizonte de tempo, ocorrerá o inverso.

A primeira bateria de testes realizados envolveu instâncias com tamanhos de 50, 100 e 150 tarefas dos tipos A e B. Estas instâncias são consideradas relativamente pequenas e, portanto, pode-se obter o valor ótimo das mesmas através de um algoritmo exato. Utilizou-se a formulação matemática aqui proposta e o *software* GLPK para obter estes resultados que foram comparados com resultados obtidos pelas versões do GRASP. Ao todo, foram testadas 50 instâncias de cada tamanho. O Gráfico 1 mostra em quantas instâncias *os resultados obtidos pelas heurísticas chegaram no valor ótimo*. O Gráfico 2 mostra, *nos casos onde o ótimo não foi alcançado*, a distância do resultado obtido pelos GRASPs em relação ao ótimo. As legendas indicam a cor das barras referentes a cada tamanho de instância.

Pelo Gráfico 1, pode-se verificar que o GRASP2, que utiliza as *Técnicas Aprimorantes* e buscas locais, consegue resultados médios bem melhores que os obtidos pela versão simples tanto nas instâncias A (sufixo "_A") quanto nas instâncias B (sufixo "_B"). Por exemplo: nas instâncias A com 50 tarefas, o GRASP2 alcançou o ótimo em 45 das 50 instâncias (90% dos casos), enquanto o GRASP1 alcançou apenas 41 ótimos dentre os 50 testados (82% dos casos). Nas instâncias B, o GRASP2 alcançou 43 ótimos (86%) contra 33 ótimos do GRASP1 (66%).

O Gráfico 2 mostra que a versão GRASP2 comparada com o GRASP1, nas instâncias A e B, também fica a uma distância menor, na média. Ou seja, quando o GRASP2 *não consegue alcançar o valor ótimo*, a solução gerada fica a uma distância menor deste valor do que a solução

gerada pelo GRASP1. A exceção fica por conta das instâncias de tamanho 50, onde o GRASP1 gera soluções com distância menor (do valor ótimo) em relação às soluções geradas pelo GRASP2, tanto nas instâncias A como nas instâncias B.

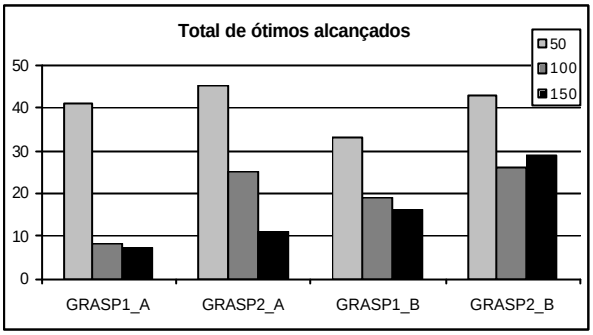

Gráfico 1: Número de ótimos alcançados

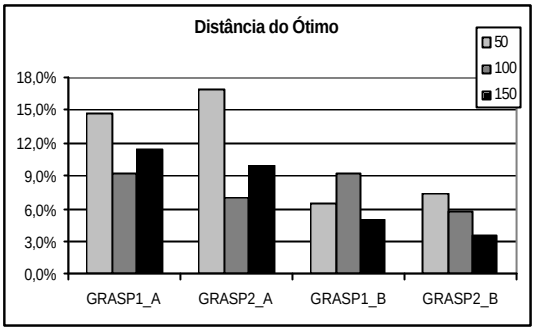

Gráfico 2: Distâncias do valor ótimo

Na tabela 1, são avaliados também instâncias de dimensões maiores (até 2000 tarefas). Na tabela 1, são mostrados os resultados médios (de três execuções) obtidos nas instâncias do tipo A. A coluna do GRASP1 mostra a distância que ele ficou do valor encontrado pelo GRASP2.

No entanto, percebemos que esta diferença não se deve às buscas locais e sim à utilização das Técnicas Aprimorantes. Para comprovar isto, são mostrados na Tabela 2, a média e o desvio padrão dos resultados gerados pela heurística ADDR, nas versões de GRASP propostas. Nesta tabela, serão mostrados apenas os resultados médios de 300 iterações da ADDR, sem nenhuma utilização de buscas locais, para as instâncias testadas. Vale ressaltar que, no GRASP1, somente a Eliminação de Arcos foi utilizada, como já foi dito, enquanto no GRASP2 foram utilizadas todas as Técnicas Aprimorantes.

Os resultados da Tabela 2 mostram uma clara superioridade do ADDR com as Técnicas em relação à versão simples. Em alguns casos, a média dos resultados do GRASP2 (ADDR com as Técnicas) é maior do que 2 vezes a média da outra versão, enquanto o desvio padrão não é tão superior assim. Isto mostra que a utilização das Técnicas aumenta muito a qualidade da solução gerada pela ADDR. Na Tabela 1 anterior, no entanto, este resultado não é tão expressivo porque o resultado final de cada GRASP é dado pelo maior valor obtido nas iterações, o que faz com que as diferenças não sejam da mesma ordem de grandeza, embora existam em favor do GRASP2 também.

Continuando a prova de que a qualidade final das soluções do GRASP depende muito mais da boa construção de uma solução inicial, a Tabela 3 mostra a média da porcentagem de contribuição de cada etapa do GRASP2. A coluna referente à Busca Local 1 teve resultados desprezíveis.

N	GRASP1	GRASP2	N	GRASP1	GRASP2	N	GRASP1	GRASP2
50a	0,00%	**47**	550a	6,50%	**8107**	1100a	0,90%	**2464**
100a	0,30%	**304**	600a	6,10%	**7177**	1200a	1,60%	**2985**
150a	0,00%	**575**	650a	3,40%	**11245**	1300a	2,20%	**2762**
200a	0,80%	**599**	700a	1,80%	**22131**	1400a	2,70%	**2393**
250a	1,60%	**1004**	750a	0,40%	**28212**	1500a	2,40%	**3743**
300a	0,60%	**1711**	800a	0,50%	**30660**	1600a	0,50%	**3298**
350a	1,50%	**1937**	850a	1,60%	**33135**	1700a	1,10%	**4467**
400a	1,60%	**5168**	900a	1,90%	**26764**	1800a	1,60%	**3918**
450a	4,80%	**6968**	950a	0,10%	**59590**	1900a	1,70%	**3926**
500a	3,40%	**11266**	1000a	0,00%	**60559**	2000a	3,80%	**5570**

Tabela 1: Diferença dos resultados médios obtidos pelos GRASPs

	GRASP1		GRASP2	
N	Média	DesvP	Média	DesvP
400a	3586,2	126,4	5112,7	42,2
800a	28056,5	1057,1	29515,7	700,9
1200a	1155,2	15,7	2898,9	16,5
1600a	1266,2	12,8	3207,9	33,8
2000a	1980,3	12,9	5422,3	56,3

Tabela 2: Média e desvio padrão dos resultados da ADDR

Técnica	ADDR	BL1	BL2
Contribuição	~99%	0%	~1%

Tabela 3: Contribuição Média de cada etapa do GRASP 2

Surgiu então a idéia de utilizar a BL3 após a BL2, para verificar se uma nova maneira de construção da solução poderia melhorar significativamente os resultados da ADDR. O que se pôde perceber é que, em determinadas instâncias, a contribuição da BL3 atingiu até 10% do valor final da solução. Estas instâncias são principalmente aquelas que contêm uma porcentagem final de tarefas ativadas bem elevada (acima de 85% do total de tarefas). Isto se explica porque, nestas instâncias, não é tão importante quais tarefas serão ativadas (já que a grande maioria delas será). Portanto, o algoritmo de recriação (que utiliza um critério de probabilidades para as tarefas) pode se beneficiar desta relaxação e fugir melhor de ótimo locais.

Além disso, a adição da BL3 tornou o GRASP proposto mais robusto: num outro experimento, foram definidos valores-alvo para cada instância testada e foi executado o GRASP2 com e sem a BL3 três vezes para cada instância, totalizando 30 execuções para instâncias do tipo A e mais 30 execuções para instâncias do tipo B. No total, o GRASP2 com a BL3 conseguiu alcançar o valor-alvo 15% a mais de vezes (9 instâncias a mais) que do que o GRASP2 sem a BL3. Mostrando que a utilização da BL3 garante um pouco de qualidade à solução final.

No entanto, esta contribuição tem um preço. Por utilizar as Buscas Locais 2 e 3, o GRASP2 mais completo chega a consumir pouco mais de 2 vezes o tempo computacional do GRASP1, na média. Testes realizados mostraram que a BL3 consome cerca de 50% do total de tempo do GRASP, a BL2 cerca de 40% e a ADDR e BL1 juntas consomem apenas 10% do tempo total. Todavia, se for utilizado o GRASP2 somente com a BL1 a diferença de tempo computacional gasto em relação ao GRASP1 é desprezível. Isto porque a utilização das Técnicas Aprimorantes pode ser feita por meio de simples cálculos algébricos, que consomem um reduzido tempo computacional, comparado com as Buscas Locais 2 e 3.

5- Conclusões

Este trabalho teve como objetivo apresentar uma nova modelagem para o problema de escalonamento de tarefas com restrições de recursos, onde a quantidade de recursos disponíveis é função direta dos recursos trazidos pela execução das tarefas.

Foram propostos também uma formulação matemática para o problema, para possibilitar a geração de soluções ótimas em pequenas instâncias, e um conjunto de Técnicas Aprimorantes que visam basicamente reduzir o tempo computacional necessário para a execução do problema e tentar uma melhoria da qualidade das soluções geradas pela heurística.

Para realizar os testes, foram propostas, inicialmente, duas versões do GRASP: uma sem as três das Técnicas e com uma Busca Local e outra com as Técnicas Aprimorantes e duas buscas locais.

Os resultados de alguns tipos de teste provaram que a utilização das Técnicas Aprimorantes melhora significativamente a qualidade da solução obtida sem comprometer o tempo computacional do algoritmo GRASP. As buscas locais, no entanto, devem ser utilizadas com muito

cuidado, pois consomem um tempo computacional grande comparado ao benefício que trazem. A exceção fica por conta da BL3 que consegue razoáveis melhorias em instâncias com uma alta porcentagem de tarefas ativadas ao final da fase de construção do ADDR.

*Pesquisa parcialmente financiada pela FAPERJ: E-26/ 150.575/2004

6- Referências Bibliográficas:

Abeyasinghe M. C. L., Greenwood, D. J. e Johansen, D. E. (2001), An efficient method for scheduling construction projects with resource constraints. *International Journal of Project Management,* 19, 29–45.

Aiex, R. et al. (2005), GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17, n. 2, 224–247.

Artigues, C., Michelon, P. e Reusser, S. (2003), Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research,* 149, 249–267.

Bastos, L. O., Ochi, L. S. e Macambira, E. M. (2005), GRASP with Path Relinking for the SONET Ring Assignment Problem. *Proc. of the 5th International Conference on Hybrid Intelligence System (HIS2005) in cooperation with IEEE Computational Intelligence Society*, 239-244.

Bouleimen, K. e Lecocq, H. (2003), A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode versions. *European Journal of Operational Research,* 149, 268–281.

Brucker, P. et al. (1999), Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research,* 12, 3–41.

Feo, T. e Resende, M. (1995), Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6, 109–133.

Foulds, L. R. e Wilson, J. M. (2005), Scheduling operations for the harvesting of renewable resources. *Journal of Food Engineering,* 70, 281–292.

Gonçalves, J., Mendes, J. e Resende, M. (2005), A random key based genetic algorithm for the resource constrained project scheduling problems. *International Journal of Production Research* (submetido).

Mika, M., Waligóra, G. e Wezglarz, J. (2005), 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,* 164, 639–668.

Senkul, P. e Toroslu, I. (2005), An architecture for workflow scheduling under resource allocation constraints. *Information Systems,* 30, 399–422.

Silva, G. C. S. et al. (2006), New heuristics for the Maximum Diversity Problem. *Journal of Heuristics* (submetido).

Souza, M. J. F., Maculan, N. e Ochi, L. S. A GRASP – TABU SEARCH algorithm for solving school timetabling problems. D. Z. Du and P. M. Pardalos (Eds.), *Metaheuristics: Computer Decision – Making,* KLUWER Academic Publishers, 659–672, 2003.