versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

UM ALGORITMO EXATO PARA O PROBLEMA DE PROGRAMAÇÃO DE PROJETOS COM CUSTO DE DISPONIBILIDADE DE RECURSOS E MÚLTIPLOS MODOS

Denise Sato Yamashita
Reinaldo Morabito *
Departamento de Engenharia de Produção
Universidade Federal de São Carlos (UFSCar)
São Carlos – SP
denisesy@dep.ufscar.br
morabito@power.ufscar.br

* *Corresponding author* / autor para quem as correspondências devem ser encaminhadas

Recebido em 10/2005; aceito em 12/2006 após 1 revisão
Received October 2005; accepted December 2006 after one revision

Resumo

O objetivo deste artigo é propor um algoritmo exato para gerar curvas de *tradeoff* entre o custo e o prazo de um projeto, baseado no problema de custo de disponibilidade de recursos com múltiplos modos de execução. Duas versões do algoritmo são propostas, a primeira é uma adaptação de um algoritmo exato da literatura, no qual só existe um modo de executar as atividades, e a segunda versão incorpora estratégias para melhorar o desempenho do método, resultando numa redução significativa de tempo computacional. Convém salientar que o algoritmo proposto é viável computacionalmente apenas para resolver problemas de tamanho moderado. As duas versões do algoritmo foram testadas resolvendo-se diversos exemplos gerados pelo programa Progen da literatura. Curvas de *tradeoff* são apresentadas e analisadas, ilustrando como o método pode ser usado em situações onde o decisor é confrontado com a difícil tarefa de balancear custos e datas de entrega do projeto.

Palavras-chave: programação de projetos; múltiplos modos de execução; custo de disponibilidade de recursos; algoritmo exato; *tradeoff* entre custos e data de entrega.

Abstract

In this paper we propose an exact algorithm to generate tradeoff curves between cost and time of a project, based on the multi-mode resource availability cost problem. Two versions of the algorithm are proposed, the first is an adaptation of an exact algorithm proposed in the literature, where there is only a single mode to execute the activities, and the second incorporates strategies in order to improve the performance of the solution method, resulting in a significant decrease in the computational time. It is worth noting that the proposed algorithm is computationally viable to solve only problems of moderate size. Both versions of the algorithm were tested solving different instances generated by the software Progen. Tradeoff curves are presented and analyzed, illustrating how the method can be used in situations where the decision maker is confronted with the difficult task of balancing costs and deadlines of a project.

Keywords: project scheduling; multiples modes; resource availability cost; exact algorithm; tradeoff between cost and deadline.

1. Introdução

Um projeto consiste basicamente de atividades que necessitam de recursos para serem concluídas, e pode aparecer em diversas áreas tais como construção civil, manufatura, telecomunicações, desenvolvimento de *software*/computação. Exemplos de projetos são construções de fábricas, pontes, estradas e prédios, manutenções em equipamentos e aeronaves, desenvolvimentos de *software* e experimentos de pesquisa, entre muitos outros (Hartmann, 2000; Pollack-Johnson & Liberatore, 1998). Na construção de uma ponte, por exemplo, a compra de material, contratação de mão-de-obra, e as diversas etapas que compõem a construção, como preparação do terreno, fundação, estruturas de pilares, vigas e lajes, pavimentação, são algumas das atividades do projeto. Essas atividades têm uma duração prevista e para serem executadas estão sujeitas a relações de precedência. Isto é, certas atividades só podem ser realizadas após outras terem sido completadas. Ao serem executadas, as atividades do projeto consomem recursos, como máquinas, mão-de-obra, dinheiro. O objetivo da programação de projetos é fazer a programação (*schedule*) das atividades de forma a otimizar um ou mais critérios (por exemplo, minimizar custos e data de entrega), obedecendo às relações de precedência entre as atividades e a disponibilidade de recursos.

As primeiras abordagens de problemas de programação de projetos (*project scheduling*) surgiram no fim da década de 50, com o desenvolvimento de métodos clássicos de caminho crítico CPM (*critical path method*) e PERT (*project evaluation and review technique*) (Moder *et al*., 1983, contém um histórico interessante). Desde então, a programação de projetos tem sido objeto de intensa pesquisa em diversas áreas. Um exemplo de aplicação pode ser visto em Takamoto *et al*. (1995), que abordam o problema de programar as atividades com o propósito de nivelar os recursos utilizados na construção civil. Gemmill & Edwards (1999) estudam o problema de manutenção de aeronaves de carga numa base da força aérea americana (Warner Robins Air Logistics Center). A manutenção destas aeronaves envolve uma grande quantidade de atividades, como desmontar e inspecionar quase todos os sistemas. O problema consiste em encontrar a melhor seqüência para completar as atividades de forma que a aeronave retorne da manutenção o mais rápido possível. Um outro exemplo prático de programação de projetos reportado na literatura aparece em experimentos de pesquisa médica realizados na Universidade de Kiel, Alemanha (Hartmann, 2000). Tais experimentos são repetidos diversas vezes, e o problema consiste em determinar a seqüência com que estes experimentos devem ser realizados, levando em conta restrições como a capacidade do laboratório, equipamento e pessoal disponíveis. Outras aplicações de programação de projetos podem ser encontradas, por exemplo, na indústria química, farmacêutica, e de papel (Neumann *et al*., 2002).

Numa pesquisa realizada por Pollack-Johnson & Liberatore (1998) com 240 empresas norte-americanas, mais de 90% dos entrevistados afirmam utilizar um *software* para gerenciamento de projetos, e 50% dos entrevistados utilizam um *software* em todos os projetos, mostrando que existe um interesse real das empresas entrevistadas por tal ferramenta. A pesquisa incluiu empresas que atuam em diversas áreas, como construção civil (18%), desenvolvimento de *software*/computação (21%), telecomunicações (18%), manufatura (17%) e outras. Segundo este estudo, cerca de 90% dos entrevistados utilizam CPM, e 62% utilizam técnicas de programação de projetos que envolvem programação ou nivelamento de recursos. A literatura sobre programação de projetos é bastante extensa; boas revisões deste tema podem ser encontradas em Brucker *et al*. (1999) e Kolisch & Padman (2001).

Dois aspectos importantes sobre programação de projetos são tratados neste artigo. Primeiro, permitir que os recursos disponíveis para o projeto sejam variáveis de decisão, sendo que existe um custo associado à disponibilidade de cada recurso. O objetivo é minimizar o custo total de alocação dos recursos que estarão disponíveis durante o projeto e, concomitantemente, definir a programação (*schedule*) das atividades a serem executadas, de modo que o projeto respeite restrições de precedência entre as atividades e termine dentro do prazo de entrega pré-estabelecido. Segundo, permitir que as atividades possam ser executadas de modos alternativos (múltiplos modos de execução). Por exemplo, uma atividade que requer 10 períodos de tempo para ser executada por 1 trabalhador, poderia alternativamente ser realizada em 4 períodos de tempo utilizando 2 trabalhadores. É interessante observar que a função dos custos a ser minimizada e a existência de uma data de entrega para o projeto induzem um equilíbrio entre o tempo e o custo do projeto: quanto mais recursos (máquinas, mão-de-obra) são alocados, mais rápido o projeto pode ser concluído, uma vez que diversas atividades podem ser realizadas simultaneamente, porém, quanto mais recursos forem alocados, mais caro será o projeto. É importante observar que, do ponto de vista da teoria de complexidade, o caso particular deste problema no qual as atividades não têm múltiplos modos de execução, denominado problema de custo de disponibilidade de recursos (PCDR – *resource availability cost problem*), é classificado como NP-difícil (Möhring, 1984).

Não temos conhecimento de artigos na literatura apresentando algoritmos exatos para resolver o problema de custo de disponibilidade de recursos com múltiplos modos de execução (PCDRMM – *multi-mode resource availability cost problem*). Möhring (1984) apresentou um algoritmo exato para resolver o caso particular PCDR, motivado por projetos de construção de pontes. Outro algoritmo exato para o PCDR foi proposto por Demeulemeester (1995). Experimentos computacionais realizados nos problemas testes de projetos de construção de pontes indicaram que este algoritmo é mais rápido do que o proposto por Möhring (1984). Rangaswamy (1998) também propôs um algoritmo exato para o PCDR baseado no método *branch-and-bound*, com bons resultados computacionais em relação aos anteriores nos mesmos problemas testes. A comparação destes resultados não é conclusiva devido a diferentes plataformas computacionais utilizadas nos experimentos. Limitantes inferiores e superiores para o PCDR foram propostos por Drexl & Kimms (2001), obtidos através de relaxação Lagrangiana e métodos de geração de colunas. Recentemente, um método heurístico foi proposto para o PCDR por Yamashita *et al*. (2006).

No presente trabalho propomos um algoritmo exato para gerar curvas de *tradeoff* entre o custo e o prazo do projeto, resolvendo-se o PCDRMM. Curvas de *tradeoff* são instrumentos de análise importantes no processo de tomada de decisão. A Figura 1 ilustra um exemplo deste tipo de curva (como neste exemplo as datas assumem valores discretos, a rigor tratam-se de pontos de *tradeoff*, ao invés de uma curva contínua de *tradeoff*). Para um dado valor de custo do projeto, o gráfico mostra a menor data de entrega relativa a este custo e vice-versa, isto é, para um dado valor de data de entrega, o menor custo para realizar este projeto dentro deste prazo. O decisor pode analisar o impacto entre escolher, por exemplo, o ponto A, que corresponde a um projeto com custo 211 e data de entrega 18, ou o ponto B, que corresponde a um projeto de custo maior, 264, porém, possui uma data de entrega menor, igual a 14.

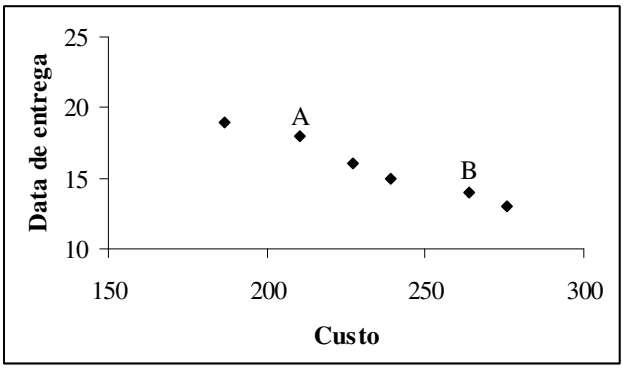

Figura 1 – Curva de *tradeoff* entre o custo e o prazo do projeto para o PCDRMM.

O algoritmo proposto é baseado no procedimento desenvolvido por Demeulemeester (1995) para resolver o PCDR. O primeiro passo do algoritmo consiste em determinar um limitante inferior para a quantidade de recursos disponíveis. Uma vez fixada a disponibilidade de recursos, um problema de programação de projetos com recursos limitados e múltiplos modos de execução para as atividades (PPPRLMM – *multi-mode resource constrained project scheduling problem*) é resolvido, com o objetivo de descobrir se existe um programa que termine dentro da data de entrega, e respeite as restrições de recurso e precedência. Se tal programa existir, esta disponibilidade de recursos é a solução ótima do PCDRMM. Caso não exista um programa que termine antes da data de entrega do projeto, então a disponibilidade de um dos tipos de recursos é aumentada em uma unidade, e o PPPRLMM relativo a esta disponibilidade de recursos é resolvido. Este processo de aumento gradual de recursos e subseqüente solução de um PPPRLMM (análise marginal) é repetido enquanto não for possível encontrar um programa que respeite a data de entrega. Para solucionar os PPPRLMMs, utilizamos um algoritmo *branch-and-bound* proposto por Sprecher & Drexl (1998), considerado um dos melhores algoritmos exatos da literatura para resolver o PPPRLMM, segundo Hartmann (2000) que o comparou a outros algoritmos *branch-and-bound* para o PPPRLMM. Este método é tratável computacionalmente apenas em situações onde é razoável considerar a disponibilidade de cada tipo de recurso em unidades inteiras, e cada disponibilidade puder ser descrita em poucas unidades (por exemplo, poucas dezenas de trabalhadores, ao invés de muitas dezenas ou centenas de trabalhadores).

É importante notar que a inclusão de múltiplos modos de execução das atividades torna o problema de programação de projetos mais realista e abrangente, mas resulta num aumento de variáveis de decisão do problema. Conseqüentemente, o problema fica mais difícil de ser resolvido do ponto de vista de otimização combinatória. Este artigo está organizado da seguinte forma: a descrição do PCDRMM e a discussão de problemas relacionados são apresentadas na seção 2, o método de resolução proposto e um exemplo ilustrativo são discutidos na seção 3, uma segunda versão do método proposto é apresentada na seção 4, os experimentos computacionais, onde são avaliados o potencial do método proposto para resolver tanto um único PCDRMM como gerar curvas de *tradeoff* entre o custo e o prazo do projeto, são descritos na seção 5, e as conclusões e perspectivas para pesquisa futura são apresentadas na seção 6.

2. O Problema de Custo de Disponibilidade de Recursos com Múltiplos Modos (PCDRMM)

Considere um projeto com n atividades em que as atividades 1 e n são atividades artificiais que indicam o início e o fim do projeto, respectivamente. O conjunto das relações de precedência entre as atividades é dado pelo conjunto H, que consiste de pares (h, j), tal que se $(h, j) \in H$, então a atividade h precede a atividade j. Cada atividade j pode ser executada em M_j modos. Quando executada no modo i, a atividade j tem duração d_{ji} e requer r_{jik} unidades do recurso do tipo k ($k = 1, ..., m$) durante a sua execução. Estes recursos são chamados renováveis porque após serem utilizados, ficam disponíveis novamente para uso no período seguinte. Exemplos de tais recursos são trabalhadores e máquinas, contra exemplos são matérias primas e dinheiro. Seja D a data de entrega do projeto, e $C_k(a_k)$ uma função de custo não decrescente associada à disponibilidade a_k do recurso do tipo k. Nos trabalhos da literatura que abordam o PCDR, os experimentos computacionais são realizados considerando $C_k(a_k)$ como uma função linear, pelo fato de utilizarem métodos de programação linear inteira. Neste artigo o mesmo se aplica, entretanto, é importante observar que o procedimento aqui proposto também pode ser utilizado (sem modificações) para problemas em que a função de custo é não-linear. As variáveis do problema são as variáveis inteiras relacionadas com a disponibilidade de recurso a_k e com a determinação dos instantes de término e modo de execução das atividades. Em certas aplicações, a disponibilidade de recurso a_k pode variar com o instante de tempo t, isto é, a_{kt}; entretanto, neste trabalho, admite-se que a variável a_k é constante com relação ao tempo. Um modelo matemático para o PCDRMM, baseado no modelo proposto por Talbot (1982) para o PPPRLMM, é dado a seguir. Neste modelo, as variáveis de decisão são dadas por:

- $x_{jit} = \begin{cases} 1, \text{ se a atividade } j = 1,...,n \text{ é executada no modo } i = 1,...,M_j \text{ e termina} \\ \text{no instante } t = 1,...,D. \\ 0, \text{ caso contrário.} \end{cases}$
- a_k = quantidade de recurso do tipo k disponível ao longo do projeto, $k = 1, ..., m$.

Os dados de entrada do problema são:

- $C_k(a_k)$ = função de custo não decrescente associada à disponibilidade a_k do recurso do tipo k.
- D = data de entrega do projeto.
- H = conjunto de relações de precedência.
- d_{ji} = duração da atividade j quando executada no modo i.
- r_{jik} = quantidade de recurso do tipo k que a atividade j utiliza quando executada no modo i.
- LF_j = instante de término mais tarde que a atividade j pode ser completada, sem violar a data de entrega do projeto, obedecendo às relações de precedência entre as atividades, sem considerar as restrições de recursos. Este valor pode ser obtido ao fazer a programação das atividades de forma regressiva, começando da última atividade artificial (n), que é programada no instante D. As demais atividades são então programadas iterativamente, sempre que todos os seus sucessores já tenham sido programados.

- EF_j = instante de término mais cedo que a atividade j pode ser completada, obedecendo às restrições de precedência entre as atividades, sem considerar as restrições de recursos. Este valor pode ser obtido ao fazer a programação das atividades de forma progressiva, começando da primeira atividade artificial (1), que é programada no instante zero. As demais atividades são então programadas o mais cedo possível, após todos os seus predecessores terem sido programados.

$$\text{Minimizar} \quad \sum_{k=1}^{m} C_k(a_k) \tag{1}$$

sujeito a

$$\sum_{i=1}^{M_j} \sum_{t=EF_j}^{LF_j} x_{jit} = 1, \qquad j = 1,...,n \tag{2}$$

$$\sum_{i=1}^{M_h} \sum_{t=EF_h}^{LF_h} t \cdot x_{hit} \leq \sum_{i=1}^{M_j} \sum_{t=EF_j}^{LF_j} (t - d_{ji}) \cdot x_{jit} \quad j = 1,...,n, \text{ e para todo } h, \text{ tal que } (h, j) \in H \tag{3}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{M_j} \sum_{b=t}^{t+d_{ji}-1} r_{jik} x_{jib} \leq a_k \qquad k = 1,...,m \quad t = 0,...,D \tag{4}$$

$$x_{jit} \in \{0,1\} \qquad j = 1,...,n, \quad t = 1,...,D, \quad i = 1,...,M_j \tag{5}$$

$$a_k \geq 0, \quad a_k \in \mathrm{Z} \qquad k = 1,...,m \tag{6}$$

Na formulação (1)-(6) acima, (1) refere-se à função objetivo do PCDRMM que consiste em minimizar o custo total de alocação de recursos do projeto. A restrição (2) assegura que cada atividade é executada exatamente uma vez, e em apenas um modo de execução. A restrição (3) garante que as relações de precedência entre as atividades são obedecidas, e a restrição (4) assegura que a soma da quantidade de recursos utilizados pelas atividades num determinado instante de tempo t não ultrapasse a quantidade de recursos disponíveis para o projeto. Finalmente, a restrição (5) define as variáveis de decisão binárias e a restrição (6) garante que a quantidade de recursos alocada seja positiva e inteira. Observe que a restrição que assegura que o projeto termine antes da data de entrega D é considerada implicitamente no modelo, uma vez que nele as variáveis binárias x_{jit} sempre têm $t \leq D$. Um exemplo ilustrativo (exemplo 1) do PCDRMM é descrito a seguir.

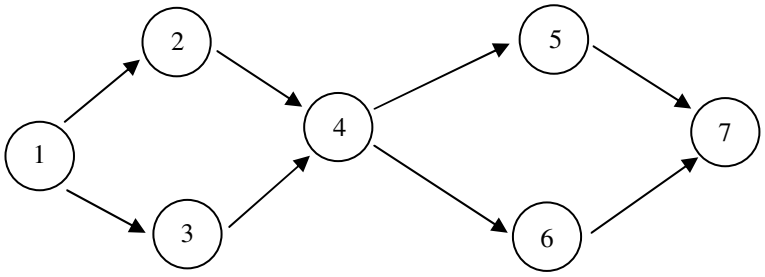

Figura 2 – Relações de precedência do projeto do exemplo 1.

Tabela 1 – Dados de entrada do exemplo 1.

Atividade j		**Modo de execução** i	**Duração** d_{ji}	**Recurso 1** r_{ji1}	**Recurso 2** r_{ji2}	**Recurso 3** r_{ji3}
1	$M_1 = 1$	1	0	0	0	0
2	$M_2 = 1$	1	5	1	2	3
3	$M_3 = 1$	1	5	1	2	3
4	$M_4 = 2$	1	5	3	1	1
		2	10	1	1	1
5	$M_5 = 1$	1	5	1	3	2
6	$M_6 = 1$	1	5	1	3	2
7	$M_7 = 1$	1	0	0	0	0

A Figura 2 mostra as restrições de precedência de um projeto com $n = 7$ atividades e $m = 3$ tipos de recursos. A Tabela 1 apresenta a duração d_{ji} e a quantidade de recursos r_{jik} utilizados por cada atividade j, quando executada no modo i. Note que a atividade 4 tem dois modos de execução para as atividades, isto é, $M_4 = 2$. Suponha que o custo do projeto seja dado por $c_1a_1 + c_2\,a_2 + c_3a_3$, no qual $c_1 = 1$, $c_2 = 5$, $c_3 = 2$, e que a data de entrega do projeto seja igual a $D = 20$. Se forem alocadas 2 unidades de recursos do tipo 1, 4 unidades de recursos do tipo 2 e 6 unidades de recursos do tipo 3 para o projeto (i.e., $a_1 = 2$, $a_2 = 4$ e $a_3 = 6$), então o custo do projeto será igual a 34. Um programa infactível para este projeto pode ser visto na Figura 3. Ela mostra, no eixo horizontal, os instantes inicial e final de cada atividade. No eixo vertical, a figura exibe a quantidade de recursos do tipo $k = 1, 2, 3$, disponíveis para o projeto. Note que a atividade 4 está sendo executada no modo 2. Observe que este programa é infactível com relação à data de entrega do projeto $D = 20$, porém factível com relação à quantidade máxima de recursos utilizados (que não pode ultrapassar 2 unidades para o recurso 1, 4 unidades para o recurso 2 e 6 unidades para o recurso 3).

A formulação (1)-(6) do PCDRMM supõe que o decisor tenha uma data de entrega D definida para o projeto, e a questão é determinar qual o projeto de menor custo que pode ser realizado dentro desta data de entrega. Alternativamente, podem existir situações em que o decisor tem um orçamento $\overline{C}$ definido para o projeto, e deseja saber qual o menor prazo para realizar o projeto (minimização do *makespan*, i.e., o instante de término da última atividade do projeto) dentro desta limitação de orçamento. O modelo (7)-(9) abaixo descreve tal caso – note que a função objetivo (7) agora minimiza a data de entrega e a restrição (9) refere-se à limitação orçamentária do projeto. Estas duas formulações (modelo (1)-(6) e modelo (7)-(9) abaixo) poderiam ser utilizadas para gerar curvas de *tradeoff* entre o custo e o prazo para finalizar um projeto. Entretanto, neste trabalho, escolhemos arbitrariamente a primeira formulação, isto é, abordamos o problema PCDRMM.

$$\text{Minimizar} \quad \sum_{t=EF_n}^{LF_n} t \cdot x_{n1t} \tag{7}$$

sujeito a (2), (3), (4), (5) e (6) (8)

$$\sum_{k=1}^{m} C_k(a_k) \le \overline{C} \tag{9}$$

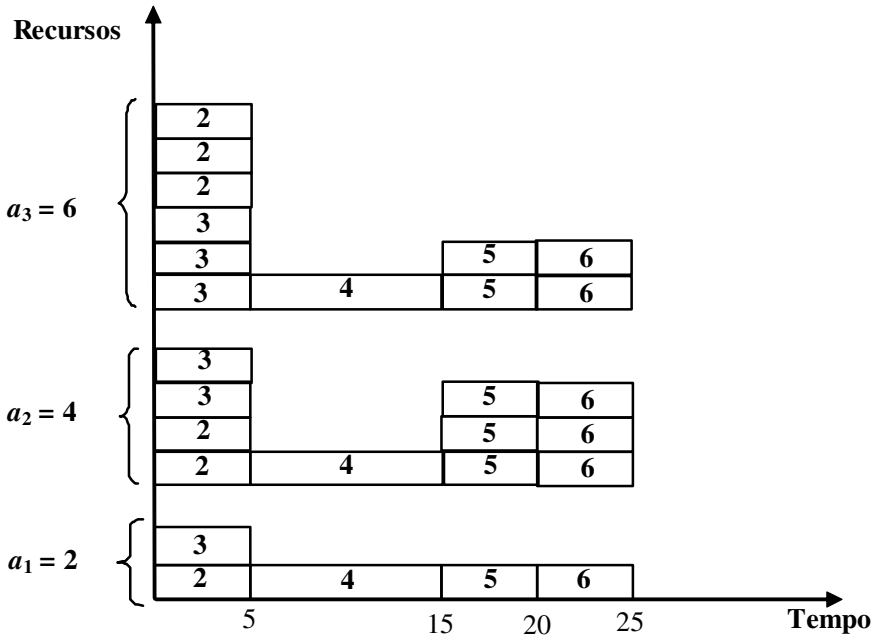

Figura 3 – Diagrama de Gantt para o projeto do exemplo 1.

3. Procedimento de Solução para o PCDRMM

Conforme mencionado, o algoritmo aqui proposto para resolver o PCDRMM (modelo (1)-(6)) é baseado no procedimento de Demeulemeester (1995) para resolver o caso particular em que não existem múltiplos modos de execução (PCDR). Como o algoritmo proposto resolve diversos PPPRLMMs ao longo de sua execução, a seguir descrevemos brevemente a formulação do PPPRLMM e o algoritmo utilizado para resolvê-lo. As principais diferenças entre o PPPRLMM e o PCDRMM são a função objetivo e a disponibilidade de recurso a_k, $k = 1,\ldots, m$. No PPPRLMM, a função objetivo é a minimização do *makespan* e a disponibilidade de recurso é um dado de entrada do problema. A formulação (10)-(11) descrita abaixo para o PPPRLMM foi proposta por Talbot (1982).

$$\text{Minimizar} \quad \sum_{t=EF_n}^{LF_n} t \cdot x_{n1t} \tag{10}$$

sujeito a (2), (3), (4) e (5). (11)

Na formulação acima, (10) refere-se à função objetivo do PPPRLMM, que consiste em minimizar o *makespan*. As demais restrições são como definidos na seção 2. Note que, diferente das formulações da seção 2, esta formulação não inclui as restrições (6), uma vez que as disponibilidades de recursos agora são parâmetros do modelo (ao invés de variáveis).

Neste trabalho, o PPPRLMM (modelo (10)-(11)) é solucionado utilizando-se um algoritmo *branch-and-bound* proposto por Sprecher & Drexl (1998), que gera uma árvore de busca cujos nós representam programações parciais, e os ramos da árvore selecionam as próximas

atividades a serem programadas e seus respectivos modos de execução. Para percorrer a árvore, utiliza-se uma busca em profundidade. Desta forma, o procedimento de enumeração começa programando a atividade artificial que marca o início do projeto no instante de tempo zero. A cada nível da árvore de busca, determina-se o conjunto das atividades que já foram programadas até então (*schedule* parcial) e o conjunto das atividades candidatas, isto é, atividades que ainda não foram programadas, mas cujos predecessores já se encontram no programa parcial. Seleciona-se, então, uma atividade do conjunto das atividades candidatas e um respectivo modo de execução. Isto gera um novo ramo da árvore de busca. Prossegue-se desta forma até que todas as atividades tenham sido programadas, e com isso, um *makespan* para o projeto é obtido. Para mais detalhes deste algoritmo, o leitor pode consultar Sprecher & Drexl (1998).

As principais diferenças entre o algoritmo de Demeulemeester (1995) e o aqui proposto consiste na utilização do algoritmo de Sprecher & Drexl (1998) para resolver o PPPRLMM, e na inclusão de um procedimento para gerar curvas de *tradeoff*. O algoritmo começa determinando um limitante inferior para a quantidade de recursos disponível. Uma vez fixada a disponibilidade de recursos, um PPPRLMM (modelo (10)-(11)) é resolvido, com o objetivo de descobrir se existe um programa que termina dentro da data de entrega e respeita as restrições de recursos e precedências. Se tal programa existir, esta disponibilidade de recursos é a solução ótima do PCDRMM. Caso contrário, cria-se um conjunto de soluções candidatas denotado por *ES*. Uma solução candidata é uma configuração possível para os parâmetros a_k, $k = 1, ..., m$, isto é, ela define uma disponibilidade de recursos. Inicialmente, o conjunto *ES* consiste de soluções candidatas geradas a partir do limitante inferior para a disponibilidade de recursos, aumentando-se marginalmente este limitante. No entanto, somente soluções candidatas que são pontos eficientes são incluídas em *ES*.

Definição. Uma solução candidata com disponibilidade de recursos $(a_1,...,a_m)$ é um ponto *eficiente* se não existe outra solução candidata com disponibilidade de recursos $(a_1',...,a_m')$ em *ES* tal que $a_k' \le a_k$ para todo $k = 1,.., m$. Por exemplo, seja *ES* ={(5,2,3), (4,5,7), (5,5,1)}, então (4,6,7) não é ponto eficiente, devido à existência do ponto $(4,5,7)$ em *ES*.

O próximo passo é resolver o PPPRLMM que corresponde ao ponto eficiente de menor custo. Se o problema for infactível (do ponto de vista da data de entrega) com esta disponibilidade de recursos, este ponto eficiente é cortado do espaço de solução e são geradas *m* soluções a partir desta solução infactível, aumentando a disponibilidade de cada um dos tipos de recursos por vez, em uma unidade. Estas novas soluções são incluídas no conjunto de soluções candidatas *ES*, caso sejam pontos eficientes, e a busca prossegue resolvendo o PPPRLMM que corresponde ao ponto eficiente mais barato. Este processo é repetido até que uma solução factível seja encontrada, isto é, o projeto termine dentro da data de entrega e nenhuma disponibilidade de recurso seja violada em qualquer instante durante a execução do projeto. Esta solução é ótima para o PCDRMM.

Um pseudocódigo resumido para este procedimento de solução proposto é descrito na Figura 4, seguido de um exemplo de aplicação do algoritmo. Um pseudocódigo mais detalhado, baseado no trabalho de Demeulemeester (1995), é apresentado no Apêndice 1. O passo 1 consiste em calcular um simples limitante inferior (b_1, ..., b_m) para a disponibilidade (a_1, ..., a_m) de recursos. O objetivo do passo 2 é refinar os limitantes inferiores (b_k) da seguinte forma: atribui-se o valor $a_k = b_k$ a um dos recursos do tipo *k*, e aos demais *m*-1 tipos de recursos *j*, $j \neq k$, atribui-se um simples limitante superior $a_j = u_j$ para a disponibilidade de

recursos, e resolve-se um PPPRLMM (procedimento *determina_*C_{max}) para esta disponibilidade de recursos (a_1, ..., a_m). Se esta solução for factível, o limitante inferior b_k não pode ser melhorado, caso contrário, o recurso, ao qual foi atribuído o valor b_k, é aumentado em uma unidade, isto é, $a_k = b_k + 1$, e um PPPRLMM (procedimento *determina_*C_{max}) é resolvido para este novo (a_1, ..., a_m). Se esta solução for factível, o procedimento *determina_*C_{max} retorna o valor 1, caso contrário, retorna o valor 0. Este processo é repetido até que uma solução factível seja encontrada. Observe que este procedimento resulta na solução ótima se tivermos apenas um tipo de recurso, isto é, $m = 1$.

O passo 3 consiste em gerar soluções candidatas variando dois tipos de recursos, o recurso do tipo 1 e um outro recurso do tipo k, $k \neq 1$. Inicialmente, estes dois tipos de recursos assumem o valor do limitante inferior $a_1 = b_1$ e $a_k = b_k$, e os demais recursos assumem valores do limitante superior. A idéia é encontrar um conjunto de duplas (a_1, a_k) que resultem num programa factível, tal que se decrescermos em uma unidade qualquer um dos recursos, a_1 ou a_k, a solução torna-se infactível. Para cada tipo de recurso k, $k \neq 1$, estes valores são armazenados nos conjuntos PES_k, que consistem de duplas (a_1, a_k) nas quais o segundo elemento especifica o número mínimo de recursos do tipo k necessários quando somente a_1 unidades estão disponíveis para o recurso do tipo 1 (onde a_1 é o primeiro elemento). O conjunto PES_k é atualizado da seguinte forma: verifica-se se já existe uma dupla em PES_k, para a qual o primeiro elemento tem o mesmo valor que a_1. Se este é o caso, o segundo elemento naquela dupla é substituído pelo valor de a_k. Caso contrário, uma nova dupla é adicionada a PES_k na qual a_1 é o valor do primeiro elemento e a_k é o valor do segundo elemento. Associado a PES_k, encontra-se o conjunto EP, que contém todos os valores de a_1 para os quais uma dupla foi adicionada a um dos conjuntos PES_k. O conjunto EP é atualizado da seguinte forma: verifica-se se EP possui um elemento cujo valor seja igual ao valor corrente a_1. Se tal elemento não existe, ele é adicionado ao conjunto. Note que se $m = 2$, o passo 3 encontra a solução ótima para o PCDRMM. Este passo está descrito detalhadamente no Apêndice 1.

O passo 4 consiste em construir o conjunto ES com soluções candidatas obtidas a partir dos elementos de PES_k. Note que essas soluções são limitantes inferiores para o problema. Para cada elemento $a_1 \in EP$, o procedimento *determina_*a_k procura as duplas em PES_k cujo primeiro elemento é o maior elemento menor ou igual a a_1, e atribui o valor do segundo elemento para a_k, $k = 1, 2,..., m$. Portanto, os valores de a_k, $k = 1, 2,..., m$, assim determinados, garantem que $a = (a_1, a_2', ..., a_m')$ é um limitante inferior para o problema quando apenas a_1 recursos do tipo 1 estão disponíveis. Verifica-se, então, se este limitante inferior pode ser incluído no conjunto ES (procedimento *inclua elemento em ES*). Se não existe elemento em ES com o mesmo valor de recursos, e se nenhum outro elemento em ES domina o elemento corrente, isto é, não existe $p = (p_1, ..., p_m)$, $p \in ES$, tal que, $p_k \leq a_k$, para todo $k = 1,..., m$, então o elemento corrente é inserido em ES. Caso contrário, nenhum elemento é colocado em ES.

No passo 5, a disponibilidade dos recursos é aquela definida pelo elemento de menor custo em ES. Se esta disponibilidade dos recursos resultar numa solução factível, então a solução ótima foi encontrada. Caso contrário, aumenta-se um dos recursos em uma unidade e o procedimento *inclua elemento em ES* é executado.

O passo 6 consiste em gerar as curvas de *tradeoff*. Neste ponto, já existe uma solução para o PCDRMM cuja data de entrega é igual a D. Se prosseguirmos explorando o conjunto ES em ordem crescente de custo, e para cada elemento de ES com mesmo valor de custo,

obtivermos o *makespan* mínimo para este valor de custo, temos então uma curva de *tradeoff* que mostra, para um dado custo, a menor data de entrega para o projeto, e vice-versa, para uma dada data de entrega, o menor custo para o projeto. O critério de parada pode ser especificado pelo decisor de várias formas, como um limite do tempo computacional, uma data de entrega mínima para o projeto, ou um custo máximo para o projeto.

Passo 1: Obter limitante inferior, b_k , para cada tipo de recurso k = 1,..., m.

Passo 2: Refinar limitante inferior, b_k , para cada tipo de recurso k = 1,..., m.

Passo 3: Para cada tipo de recurso k = 2,..., m, examinar combinações (a_1, a_k) e criar conjuntos PES_k e EP.

Passo 4: Criar conjunto ES a partir do conjunto PES_k.

Passo 5: Selecionar uma solução candidata a de menor custo em ES. Se a solução for factível, então seguir para o passo 6, pois a solução ótima foi encontrada. Caso contrário, gerar m soluções candidatas, cada uma delas geradas a partir de a aumentando-se um dos recursos em uma unidade. Substituir a pelas m soluções candidatas executando o procedimento *inclua elemento em ES*. Repetir o Passo 5.

Passo 6: Fase 2: curva de *tradeoff*. Examinar elementos de ES (em ordem não-decrescente de custo) e dentre as soluções de mesmo custo, encontrar a de menor *makespan*. Guardar o valor do custo e seu respectivo *makespan* mínimo. Para cada solução de ES examinada, aumentar um dos recursos em uma unidade e executar o procedimento *inclua elemento em ES*. Repetir este passo até que critério de parada tenha sido satisfeito.

Figura 4 – Pseudocódigo do procedimento proposto.

Uma prova da correção do algoritmo é apresentada no Apêndice 2. A seguir, por ilustração, o algoritmo é aplicado para resolver o exemplo 1 descrito na Seção 2.

Passo 1: O valor do limitante inferior calculado neste passo é $b_1 = 1,\ b_2 = 3,\ b_3 = 3$.

Passo 2: O valor do limitante superior é a soma dos recursos, e resulta em $u_1 = 7$, $u_2 = 11$ e $u_3 = 11$. O objetivo deste passo é melhorar o limitante inferior. Inicialmente, atribui-se o valor do limitante inferior ao recurso do tipo 1, e o valor do limitante superior aos demais tipos de recursos, isto é, a solução a ser avaliada é $(a_1, a_2, a_3) = (1,11,11)$. Ao resolver o PPPRLMM para estas disponibilidades de recursos, obtemos um valor para o *makespan* que excede a data de entrega. Assim, o recurso 1 é aumentado em uma unidade e determina-se o *makespan* para $(a_1, a_2, a_3) = (2,11,11)$. Esta solução é factível, e portanto o novo limitante inferior para o recurso do tipo 1 é $b_1 = 2$. Este procedimento para determinar o limitante inferior é repetido para o recurso do tipo 2: analisa-se (7,3,11), que gera uma solução infactível e portanto, é preciso aumentar mais uma unidade do recurso do tipo 2, gerando (7,4,11), que resulta numa solução factível, e assim, o novo limitante inferior para o recurso do tipo 2 é $b_1 = 4$. Prosseguindo da mesma forma para o recurso do tipo 3, temos $b_3 = 4$. Portanto, ao final do Passo 2 temos que o novo valor do limitante inferior para cada tipo de recurso é $b_1 = 2,\ b_2 = 4,\ b_3 = 4$.

Passo 3: Começando com o recurso do tipo 2, $k = 2$, enumeram-se todos os pares (a_1, a_2), que resultem numa solução factível, tal que o decréscimo de um destes recursos, em uma unidade, torna a solução infactível, isto é, $(a_1\text{-}1, a_2, a_3)$ e $(a_1, a_2\text{-}1, a_3)$ sejam infactíveis. Seguindo o algoritmo descrito no Apêndice 1, parte-se da solução $(a_1, a_2, a_3) = (b_1, b_2, u_3) = (2, 4, 11)$. O recurso a_2 é aumentado em uma unidade de cada vez, até atingir o valor $a_2 = 6$, que resulta numa solução factível. Atualizando PES_2, temos: $PES_2 = \{(2,6)\}$ e $EP = \{2\}$. A disponibilidade de recurso a_1 é aumentada em uma unidade, $a_1 = 2+1 = 3$, e o valor de a_2 é reduzido em uma unidade até que solução torne-se infactível ou um limitante inferior para o recurso seja atingido. Desta forma, examinam-se:

- (3,5,11) – solução factível, $PES_2 = \{(2,6),(3,5)\}$ e $EP = \{2,3\}$.
- (3,4,11) – solução factível, $PES_2 = \{(2,6),(3,4)\}$ e $EP = \{2,3\}$.

Como $a_2 = b_2$, então a_2 não pode ser reduzido mais porque atingiu o valor do limitante inferior. Este processo é repetido para $k = 3$:

- (2,11,4) – solução infactível, então se aumenta o recurso 3 em uma unidade.
- (2,11,5) – solução infactível, então se aumenta o recurso 3 em uma unidade.
- (2,11,6) – solução factível, $PES_3 = \{(2,6)\}$ e $EP = \{2,3\}$ (2 já está em *EP*).

O procedimento prossegue como descrito no algoritmo e resulta em $PES_3 = \{(2,6), (3,4)\}$ e $EP = \{2,3\}$.

Passo 4: O primeiro elemento em *EP* é 2, e do conjunto PES_2 temos $a_2 = 6$, e de PES_3, temos $a_3 = 6$, portanto, $ES = \{(2,6,6)\}$ com custo: $1 \cdot 2 + 5 \cdot 6 + 2 \cdot 6 = 44$. O segundo elemento em *EP* é 3, logo, $ES = \{(3,4,4), (2,6,6)\}$, o custo de (3,4,4) é: $1 \cdot 3 + 5 \cdot 4 + 2 \cdot 4 = 31$. Note que *ES* está ordenado em ordem crescente de custo.

Passo 5: Seleciona-se o primeiro elemento de *ES*, (3,4,4). Como esta solução é infactível, então os recursos são aumentados, gerando (veja Figura 5): (4,4,4,**32**), (3,5,4,**36**), (3,4,5,**33**), no qual os três primeiros números correspondem às disponibilidades de recursos, e o número em negrito corresponde ao custo do projeto, referente a esta solução. Como estes elementos ainda não foram incluídos em *ES*, e estes elementos não são dominados por nenhum outro elemento de *ES*, então estas soluções são incluídas em $ES = \{(4,4,4,\mathbf{32}), (3,4,5,\mathbf{33}), (3,5,4,\mathbf{36}), (2,6,6,\mathbf{44})\}$. O próximo elemento a ser examinado é (4,4,4,**32**), o qual resulta num programa infactível quando o PPPRLMM é resolvido. Geram-se então as soluções: (5,4,4,**33**), (4,5,4,**37**), e (4,4,5,**34**). A solução (4,5,4,**37**) é dominada por (3,5,4,**36**) e a solução (4,4,5,**34**) é dominada por (3,4,5,**33**) e portanto, não são incluídas em $ES = \{(3,4,5,\mathbf{33}), (5,4,4,\mathbf{33}), (3,5,4,\mathbf{36}), (2,6,6,\mathbf{44})\}$. Seguindo a ordem de *ES*, examina-se (3,4,5,**33**), que resulta numa solução infactível, e as seguintes soluções são geradas (4,4,5,**34**), (3,5,5,**38**) (não é incluída porque não é ponto eficiente) e (3,4,6,**35**), e portanto, $ES = \{(5,4,4,\mathbf{33}), (4,4,5,\mathbf{34}), (3,4,6,\mathbf{35}), (3,5,4,\mathbf{36}), (2,6,6,\mathbf{44})\}$. Prosseguindo o algoritmo, determina-se o *makespan* para (5,4,4,**33**), que resulta numa solução infactível, e o conjunto *ES* é atualizado, $ES = \{(4,4,5,\mathbf{34}), (6,4,4,\mathbf{34}), (3,4,6,\mathbf{35}), (3,5,4,\mathbf{36}), (2,6,6,\mathbf{44})\}$. Examina-se, em seguida, (4,4,5,**34**), que resulta numa solução infactível, e o novo conjunto *ES* é $ES = \{(6,4,4,\mathbf{34}), (3,4,6,\mathbf{35}), (5,4,5,\mathbf{35}), (3,5,4,\mathbf{36}), (2,6,6,\mathbf{44})\}$. O algoritmo prossegue determinando o *makespan* para a próxima solução de menor custo, (6,4,4,**34**), que também resulta numa solução infactível, e $ES = \{(3,4,6,\mathbf{35}), (5,4,5,\mathbf{35}), (7,4,4,\mathbf{35}), (3,5,4,\mathbf{36}), (6,5,4,\mathbf{39}), (2,6,6,\mathbf{44})\}$. Finalmente, a solução (3,4,6,**35**) tem um programa que termina antes da data de entrega do projeto e portanto, esta é a solução ótima do exemplo 2.

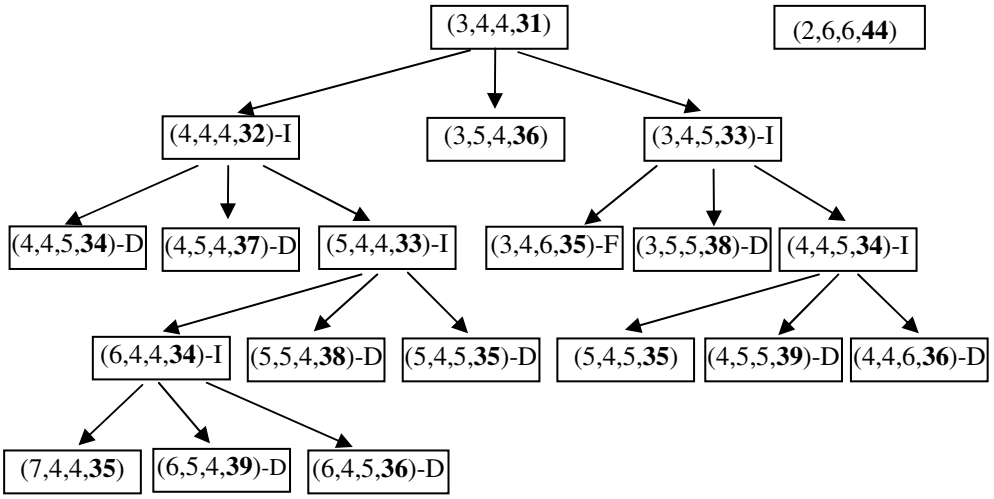

Figura 5 – Ilustração do Passo 5 do algoritmo: soluções infactíveis são marcadas com a letra "I", soluções dominadas são marcadas com a letra "D".

Passo 6: Curva de *tradeoff*: A solução (3,4,6,**35**) tem um programa que termina no instante 20, logo, *Melhor_custo* = 35 e *Melhor_makespan* = 20. O conjunto *ES* tem os seguintes valores: *ES* = {(5,4,5,**35**), (7,4,4,**35**), (3,5,4,**36**), (4,4,6,**36**), (3,4,7,**37**), (6,5,4,**39**), (6,5,6,**40**), (2,6,6,**44**)}. O algoritmo prossegue determinando o *makespan* para a próxima solução de menor custo, (5,4,5,**35**), que resulta em 25 que é maior que *Melhor_makespan*, e portanto, não é atualizado, e *ES* = {(7,4,4,**35**), (3,5,4,**36**), (4,4,6,**36**), (6,4,5,**36**), (3,4,7,**37**), (6,5,4,**39**), (6,5,6,**40**), (2,6,6,**44**)}. Finalmente, a solução (7,4,4,**35**) é examinada, e resulta num programa cujo *makespan* é igual a 25, e (8,4,4,36) é incluída em *ES*. O próximo elemento de *ES* a ser examinado tem custo 36, portanto, o valor *Melhor_custo* = 35 e *Melhor_makespan* = 20 é um ponto da curva de *tradeoff*. Este passo é repetido até que o critério de parada tenha sido alcançado.

4. Modificações do Algoritmo Proposto

Propomos a seguir duas modificações no procedimento descrito na seção 3 e na Figura 4, com o objetivo de reduzir o esforço computacional requerido.

Modificação 1. Trata-se de obter um limitante superior para o PCDRMM nos Passos 2 e 3, e utilizar este limitante para reduzir o número de elementos do conjunto *ES*. Nos Passos 2 e 3 do algoritmo, soluções factíveis para o PCDRMM são encontradas. Seja v_k a maior quantidade de recursos de fato utilizada num programa factível obtido pelo procedimento *determina_*C_{max}. Então o limitante superior para o custo pode ser escrito como $LS = \sum_{k=1}^{m} C_k(v_k)$. Este limitante superior pode ser usado para reduzir o número de elementos do conjunto *ES*, pois se a solução candidata a ser inserida no conjunto *ES*, no Passo 4 do algoritmo, tiver um valor de custo maior do que *LS*, então, esta solução candidata, certamente, não pode ser uma solução ótima para o problema. O exemplo a seguir ilustra a modificação 1.

Exemplo: No Exemplo 1, ao aplicar o Passo 2, uma solução factível é encontrada pelo procedimento *determina_*C_{max} para a disponibilidade de recursos (2,11,11). Ao examinarmos esta solução, verificamos que durante o horizonte de tempo de execução do projeto, os recursos $a_2 = 11$ e $a_3 = 11$ nunca são utilizados até o seu limite, isto é, existe uma folga na utilização destes recursos. De fato, o número máximo de recursos do tipo 2 e do tipo 3 utilizados durante o projeto foram $v_2 = 6$ e $v_3 = 6$, gerando um custo de $1 \cdot 2 + 5 \cdot 6 + 2 \cdot 6 = 44$. Portanto, 44 é um limitante superior para o custo do projeto. O mesmo raciocínio é aplicado sempre que uma solução factível é encontrada pelo procedimento *determina_*C_{max}, e o melhor limitante superior é guardado. Para o Exemplo 2, o melhor limitante superior encontrado é *LS* = 35. Se este limitante superior for aplicado aos Passos 4 e 5 do algoritmo, todos os elementos de *ES* que tiverem valor de função objetivo maior do que 35 poderiam ser eliminados de *ES*, gerando uma economia em termos de memória e esforço computacional.

Modificação 2. Esta modificação é acrescentada ao Passo 5 e envolve o procedimento *determina_*C_{max}. O procedimento *determina_*C_{max} cria uma árvore de enumeração tal que, em cada nó da árvore, uma atividade, cujos predecessores já foram programados em níveis anteriores da árvore, e um respectivo modo de execução, são selecionados. A atividade selecionada deve começar a ser executada, o mais cedo possível, respeitando as restrições de recursos e precedências. Seja *t* o instante de tempo mais cedo, no qual todos os predecessores da atividade já tenham terminado de ser executados. Um recurso *k* é *crítico*, se durante a execução do procedimento *determina_*C_{max}, existir ao menos uma atividade, que comece a ser executada no instante $S > t$, devido à insuficiência de recurso do tipo *k* no instante *t*, isto é, a atividade precisou ser atrasada porque não havia recurso *k* em quantidade suficiente. Suponha que, no Passo 5, o procedimento *determina_*C_{max} seja aplicado ao vetor de recursos $a = (a_1, a_2, ..., a_k, ..., a_m)$, e que o programa obtido por *determina_*C_{max} seja infactível. Se o recurso do tipo *k* não é crítico, então qualquer solução tal que $a' = (a'_1 = a_1, a'_2 = a_2, ..., a'_k > a_k, ..., a'_m = a_m)$ não pode ser solução ótima. A explicação é que se o recurso *k* não é crítico, então ele não influi na determinação do instante de término do projeto, e o aumento da disponibilidade deste recurso não reduzirá o *makespan*. Portanto, no Passo 5, se *a* é infactível, e o recurso *k* não é crítico, então *a'* não deve ser incluído em *ES*. O exemplo a seguir ilustra a modificação 2.

Exemplo: No exemplo 1, o primeiro elemento de *ES* é a solução (3,4,4,**31**) que resulta num programa infactível, e portanto gera os seguintes candidatos para *ES*: (4,4,4,**32**), (3,5,4,**36**), (3,4,5,**33**). Aplicando a Modificação 2, verifica-se que o recurso do tipo 1 nunca é crítico quando o procedimento *determina_*C_{max} é aplicado a (3,4,4,**31**). Portanto, a solução (4,4,4,**32**) não é incluída em *ES*, e *ES* = {(3,4,5,**33**), (3,5,4,**36**), (2,6,6,**44**)}. Seguindo a ordem de *ES*, examina-se (3,4,5,**33**), que resulta numa solução infactível, e as seguintes soluções são geradas (4,4,5,**34**), (3,5,5,**38**) e (3,4,6,**35**). A solução (3,5,5,**38**) não é incluída em *ES*, pois não é ponto eficiente, uma vez que é dominada por (3,5,4,**36**). Verifica-se que o recurso do tipo 1 não é crítico, e portanto, não é incluído em *ES*, e *ES* = {(3,4,6,**35**), (3,5,4,**36**), (2,6,6,**44**)}. Prosseguindo o algoritmo, a próxima solução examinada é (3,4,6,**35**), que resulta num programa factível, e portanto é a solução ótima do exemplo proposto.

5. Experimentos Computacionais

Para analisar o desempenho do algoritmo proposto nas seções 3 e 4, diversos experimentos computacionais foram realizados utilizando um microcomputador PC Pentium 4, 2,6 GHz, 512 Mbyte RAM. Todos os códigos foram implementados na linguagem C++. Os problemas testes para o PCDRMM (modelo (1)-(6)) foram gerados pelo programa Progen (Kolisch & Sprecher, 1996), que é um gerador de problemas testes amplamente utilizado na literatura para problemas de programação de projetos com recursos limitados. Dois importantes parâmetros do Progen são a complexidade de rede (*network complexity – NC*) e o fator de recursos (*resource factor – RF*). A complexidade da rede *NC* reflete o número médio de sucessores imediatos de uma atividade. O fator de recursos varia entre [0,1] e reflete a densidade dos diferentes tipos de recursos que uma atividade requer para ser processada. Por exemplo, se $RF = 1$, então cada atividade requer todos os m tipos de recursos, enquanto se $RF = 0$, as atividades não precisam de qualquer tipo de recurso. É também necessário determinar uma data de entrega para o projeto. Drexl & Kimms (2001) sugeriram calcular a data de entrega para o projeto como uma função do caminho crítico do projeto: $D = DF \max \{EF_n,\}$, no qual DF é o fator de data de entrega do projeto e EF_n é o instante de término mais cedo do projeto. Para cada problema teste, os custos c_k são gerados por uma distribuição uniforme U(0, 10]. Os valores de parâmetros utilizados para gerar os problemas testes foram:

- $n = 10$, $m = 4$, e $M_j = 3$ ($j = 1,..,n$).
- *RF*: 0,25, 0,5, 0,75 e 1,0.
- *NC*: 1,5, 1,8 e 2,1.
- *DF:* 1,0, 1,1, 1,2, 1,3 e 1,4.

Para cada combinação de *RF*, *NC* e *DF*, foi gerado um problema teste, totalizando 4x3x5 = 60 problemas testes. As duas versões do algoritmo proposto foram testadas: a versão 1 refere-se ao procedimento descrito na Figura 4 da seção 3, e a versão 2 inclui as duas modificações propostas na seção 4. Os testes computacionais foram realizados resolvendo-se o PCDRMM inicialmente sem gerar curvas de *tradeoff*. O algoritmo foi executado até que a solução ótima fosse encontrada. A Tabela 2 compara os tempos computacionais (em segundos) das duas versões, de acordo com o fator de data de entrega *DF*. Em geral, os tempos computacionais da versão 2 são significativamente menores do que os tempos computacionais da versão 1, mostrando que, de fato, a inclusão das duas modificações propostas são bastante eficazes. A única exceção são os problemas com $DF = 1,0$, nos quais, o custo computacional de aplicar as modificações piora o desempenho do algoritmo. Note que, nas duas versões do algoritmo proposto, os tempos computacionais são menores para problemas com data de entrega mais apertada, e maiores, para fatores de data de entrega intermediários, $DF = 1,3$ e $DF = 1,4$. Este aumento do tempo computacional se deve, provavelmente, ao aumento do número de PPPRLMMs resolvidos, como pode ser visto na Tabela 3, que mostra o número de PPPRLMMs resolvidos, em média, pelas versões 1 e 2, de acordo com o fator de data de entrega.

Tabela 2 – Tempos computacionais médios (em segundos) dos 60 problemas testes, em função do fator de data de entrega *DF*.

Versão	***DF***				
	1,0	1,1	1,2	1,3	1,4
1	3,3	25,9	87,7	93,7	97,1
2	4,7	15,1	38,7	44,0	36,9
Diferença (%)	-42,42	41,69	55,87	53,04	62,0

Tabela 3 – Número médio de problemas PPPRLMMs resolvidos nos 60 problemas testes, em função do fator de data de entrega *DF*.

Versão	***DF***				
	1,0	1,1	1,2	1,3	1,4
1	118,3	1948,8	1762,1	2157,0	1978,3
2	112,8	999,2	448,0	741,3	629,3
Diferença (%)	4,64	48,72	74,57	65,63	68,19

A Figura 6 apresenta uma análise da variação do custo do projeto em função do fator de data de entrega *DF*, para os 60 problemas testes. Nesta figura, os problemas testes são agrupados por valor de *RF*, e o gráfico mostra a variação do custo médio para cada valor de *DF*. Note que o custo do projeto cai à medida que *DF* cresce, como esperado. Para os problemas testes analisados, esta redução é mais significativa para os valores *DF* = 1,0 e *DF* = 1,1, e menos significativa para *DF* = 1,3 e *DF* = 1,4. Por exemplo, enquanto a data de entrega aumenta em 40%, o custo cai 45%, 33%, 29% e 26% para *RF* = 0,25, *RF* = 0,50, *RF* = 0,75 e *RF* = 1,00, respectivamente. Note também que o custo médio do projeto cresce à medida que *RF* cresce – isso ocorre porque quanto maior for o valor de *RF*, maior é a quantidade de tipos de recursos que as atividades necessitam para serem executadas e, portanto, é preciso ter uma quantidade mais alta de recursos disponíveis durante a execução do projeto.

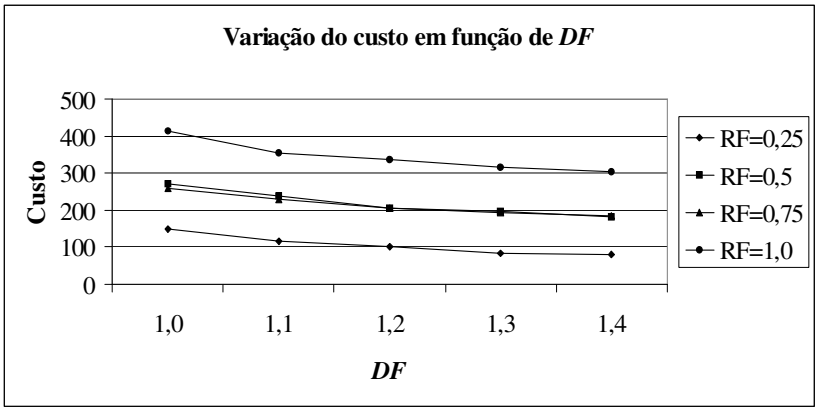

Figura 6 – Variação do custo do projeto em relação ao fator de data de entrega *DF*.

Para analisar o desempenho do algoritmo proposto para a geração das curvas de *tradeoff* entre o custo e o prazo do projeto, utilizou-se a versão 2 e 4x3 = 12 problemas testes gerados da mesma maneira que antes, variando-se RF = 0,25, 0,5, 0,75, 1,0 e NC = 1,5, 1,8, 2,1, e fixando-se DF = 1,6. O algoritmo proposto foi executado até que a data de entrega mínima fosse igual ao instante de término mais cedo do projeto. A Figura 7 exibe uma curva de *tradeoff* entre a data de entrega e custo do projeto, obtida para um destes problemas com n = 10, m = 4, M_j = 3, RF = 0,5, NC = 1,8, DF = 1,6 e data de entrega máxima igual a 17. Para um dado valor de custo do projeto, o gráfico mostra a menor data de entrega relativa a este custo e vice-versa, isto é, para um dado valor de data de entrega, o menor custo para realizar este projeto dentro deste prazo.

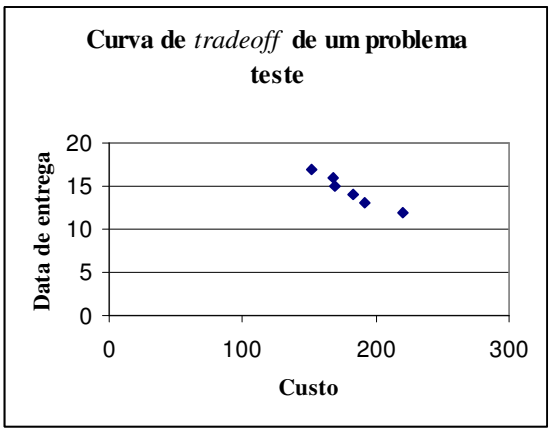

Figura 7 – Curva de *tradeoff* entre o custo e a data de entrega para um problema teste com n = 10, m = 4, M_j = 3, RF = 0,5, NC = 1,8, DF = 1,6 e data de entrega máxima igual a 17.

O tempo computacional médio gasto para gerar curvas de *tradeoff* para os problemas testes, utilizando a versão 2 do algoritmo proposto, foi de 852 segundos, em média. Com o objetivo de avaliar melhor o desempenho deste algoritmo, o modelo (1)-(6) foi implementado na linguagem de modelagem GAMS (versão 2.0.10.0) (Brooke *et al.*, 1997) e resolvido pelo *solver* CPLEX (versão 7.0). O tempo computacional médio gasto pelo aplicativo GAMS/CPLEX para gerar as mesmas curvas de *tradeoff* foi de 964 segundos, em média, isto é, 13,14% maior do que o tempo computacional do algoritmo proposto. A Tabela 4 compara os tempos computacionais, em segundos, do algoritmo proposto e do aplicativo GAMS/CPLEX para cada um dos 12 problemas testes. Note que em 8 dos 12 problemas testes, o algoritmo proposto teve um desempenho melhor do que o aplicativo GAMS/CPLEX. Observe também que o maior tempo computacional da Tabela 4 corresponde ao problema teste 7, resolvido pelo método proposto.

Tabela 4 – Tempo computacional gasto para gerar as curvas de *tradeoff* nos 12 problemas testes.

Problema teste	Método proposto	GAMS/CPLEX
1	80,84	138,54
2	748,13	163,75
3	225,36	768,35
4	346,14	3417,62
5	53,84	101,92
6	215,73	265,27
7	5700,62	429,38
8	1777,20	3119,80
9	1,06	54,17
10	32,88	415,98
11	778,34	555,02
12	268,88	2140,51
Média	852,42	964,19

6. Conclusões

Neste trabalho foi proposto um algoritmo exato para gerar curvas de *tradeoff* entre o custo e o prazo de um projeto, baseado na solução do problema PCDRMM (modelo (1)-(6)). Convém salientar que não temos conhecimento de artigos na literatura apresentando algoritmos exatos para resolver o PCDRMM. O algoritmo aqui proposto é uma extensão do algoritmo exato apresentado em Demeulemeester (1995) para o problema PCDR (modelo (1)-(6) sem múltiplos modos). A consideração de múltiplos modos de execução das atividades torna o problema de programação de projetos mais realista e abrangente, mas resulta num aumento de variáveis de decisão do problema.

Regras para limitar o número de soluções avaliadas foram desenvolvidas e acrescentadas ao algoritmo para melhorar seu desempenho computacional. Testes usando exemplos gerados pelo programa Progen da literatura mostraram que a utilização destas regras gera uma economia de tempo de computação significativa, principalmente para problemas com prazos mais folgados. Os resultados mostraram que o algoritmo proposto é viável computacionalmente para gerar curvas de *tradeoff* entre o custo e o prazo do projeto em problemas de tamanho moderado. Em particular, este algoritmo é tratável computacionalmente apenas em situações onde é razoável considerar a disponibilidade de cada tipo de recurso em unidades inteiras razoavelmente pequenas (por exemplo, poucas dezenas de trabalhadores, ao invés de muitas dezenas ou centenas de trabalhadores).

Como a literatura não dispõe de resultados de outros métodos para comparação, utilizou-se o aplicativo GAMS/CPLEX para resolver problemas PCDRMM (modelos (1)-(6)) e gerar curvas de *tradeoff* para os mesmos problemas testados com o algoritmo proposto. Este algoritmo foi em média cerca de 13% mais rápido que o aplicativo GAMS/CPLEX, e em 8 dos 12 problemas teste, foi mais rápido do que o aplicativo GAMS/CPLEX. Embora esse ganho não seja tão expressivo, acreditamos que o algoritmo proposto é um bom ponto inicial de pesquisa para o desenvolvimento de algoritmos exatos mais eficientes. Uma perspectiva importante para pesquisa futura é desenvolver heurísticas para o PCDRMM para resolver problemas de maior porte, e estender o método proposto para tratar problemas com recursos não-renováveis.

Agradecimentos

Os autores agradecem aos três revisores anônimos pelos úteis comentários e sugestões, em particular a um dos revisores (revisor 1) cujas sugestões melhoraram a apresentação e clareza do artigo. Esta pesquisa contou com o apoio do CNPq (processos 150337/2004-3 e 522973/95-4).

Referências Bibliográficas

(1) Brooke, A.; Kendrik, D. & Meeraus, A. (1997). *GAMS, Sistema Geral de Modelagem Algébrica*. Edgard Blucher Ltda.

(2) Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K. & Pesch, E. (1999). Resource constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, **112**(1), 3-41.

(3) Demeulemeester, E. (1995). Minimizing Resource Availability Costs in Time-Limited Project Networks. *Management Science*, **41**(10), 1590-1598.

(4) Drexl, A. & Kimms, A. (2001). Optimization guided lower and upper bounds for the resource investment problem. *Journal of the Operational Research Society*, **52**(3), 340-351.

(5) Gemmill, D.D. & Edwards, M.L. (1999). Improving resource-constrained project scheduling with look-ahead techniques. *Project Management Journal*, **30**(3), 44-55.

(6) Hartmann, S. (2000). Project scheduling under limited resources: models, methods, and applications. *Lecture Notes in Economics and Mathematical Systems 478*, Springer-Verlag.

(7) Kolisch, R. & Padman, R. (2001). An integrated survey of deterministic project scheduling. Omega: *The International Journal of Management Science*, **29**(3), 249-272.

(8) Kolisch, R. & Sprecher, A. (1996). PSPLIB – A project scheduling library. *European Journal of Operational Research*, **96**, 205-216.

(9) Moder, J.J.; Phillips, C.R. & Davis, E.W. (1983). *Project Management with CPM, PERT and Precedence Diagramming*. 3rd edition, Van Nostrand, New York.

(10) Möhring, R.F. (1984). Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Operations Research*, **32**(1), 89-120.

(11) Neumann, K.; Schwindt, C. & Zimmermann, J. (2002). *Project scheduling with time windows and scarce resources*. Springer-Verlag.

(12) Pollack-Johnson, B. & Liberatore, J.M. (1998). Project management software usage patterns and suggested research directions for future developments. *Project Management Journal*, **29**(2), 19-28.

(13) Rangaswamy, B. (1998). Multiple Resource Planning and Allocation in Resource-Constrained Project Networks. Ph.D. thesis, Graduate School of Business, University of Colorado.

(14) Sprecher, A. & Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, **107**(2), 431-450.

(15) Takamoto, M.; Yamada, N.; Kobayashi, Y. & Nonaka, H. (1995). Zero-one quadratic programming algorithm for resource levelling of manufacturing process schedules. *Systems and Computers in Japan*, **26**(10), 2075-2082.

(16) Talbot, F.B. (1982). Resource-constrained project scheduling problem with time-resource tradeoffs: The nonpreemptive case. *Management Science*, **28**(10), 1197-1210.

(17) Yamashita, D.S.; Armentano, V.A. & Laguna, M. (2006). Scatter search for project scheduling with resource availability cost, Special Issue on Scatter Search. *European Journal of Operational Research*, **169**(2), 623-637.

Apêndice 1. Pseudocódigo do procedimento proposto.

Passo 1: Para cada tipo de recurso $k = 1,...,m$\{
Para cada atividade $j = 1, ...,n$\{
$\lambda_{kj} = \min_{i=1,...,M_j}\{r_{jik}\}$
\}
$b_k = \max_{j=1,...,n}\{\lambda_{kj}\}$
\}

Passo 2: Para cada tipo de recurso $k = 1, 2,...,m$\{
$u_k = \sum_{j=1}^{n} \max_{i=1,...,M_j}\{r_{jik}\}$;
\}
Para cada tipo de recurso $k = 1,...,m$\{
Para cada tipo de recurso $g = 1,...,m$
$a_g = u_g$;
$sol = 0$;
Enquanto *sol* for igual a 0\{
$a_k = b_k$;
sol = *determina_*C_{max} ;
Se *sol* for igual a 0
$b_k = b_k + 1$;
\};
\};

Passo 3: $EP = \varnothing$;
Para cada tipo de recurso $k = 2,...,m$\{
$PES_k = \varnothing$;
Para cada tipo de recurso $j = 2,...,m$
$a_j = u_j$;

$a_1 = b_1$;
$a_k = b_k$;
$sol = 0$;
Enquanto *sol* igual a 0{
    sol = *determina_*C_{max};
    Se *sol* igual a 0
        $a_k = a_k + 1$;
};
atualize PES_k;
atualize EP;
$a_k = a_k - 1$;
$a_1 = a_1 + 1$;
Enquanto $a_k \geq b_k$ {
    Se *determina_*C_{max} for igual a 1{
        atualize PES_k;
        atualize EP;
        $a_k = a_k - 1$;
    }caso contrário
        $a_1 = a_1 + 1$;
};
};

Passo 4: *ES* = {};
Para cada elemento a_1 em *EP*{
    Para cada tipo de recurso $k = 2,...,m$
        a_k = *determina_*$a_k(a_1, k)$;
    inclua solução candidata $(a_1, a_2, ..., a_m)$ em *ES*;
}

Passo 5: Para cada elemento $(a_1, a_2, ..., a_m)$ em *ES*, examinado em ordem não-decrescente de custo{
    Remova elemento $(a_1, a_2, ..., a_m)$ de *ES*
    Se *determina_*C_{max} for igual a 1{
        Se deseja gerar curva de *tradeoff* {
            Para cada tipo de recurso $k = 1,...,m$ {
                Inclua solução candidata $(a_1, a_2, ..., a_m)$ em *ES* (procedimento *inclua elemento em ES*)
            }
            Melhor_custo = custo da solução ótima encontrada;
            Melhor_makespan = valor do *makespan* encontrado por *determina_*C_{max};
            Inicie fase 2 – vá para o **Passo 6**;

}caso contrário, pare com a solução ótima;
}caso contrário{
Para cada tipo de recurso $k = 1,...,m$\{
Inclua solução candidata $(a_1, a_2, ..., a_m)$ em *ES* (procedimento *inclua elemento em ES*)
}
}
}

Passo 6: Fase 2: curva de *tradeoff*.
Para cada elemento em *ES*, examinado em ordem não-decrescente de custo{
obter disponibilidade de recurso;
Obtenha makespan resolvendo um PPPRLMM;
Se custo > Melhor_custo{
Guardar Melhor_custo e Melhor_makespan como pontos da curva de *tradeoff* ;
Melhor_makespan = *makespan*;
Melhor_custo = custo;
}caso contrário{
Se *makespan* < Melhor_makespan
Melhor_makespan = *makespan*;
};
Se critério de parada não foi satisfeito{
Para cada tipo de recurso $k = 1,...,m$\{
obter disponibilidade de recurso;
$a_k = a_k + 1$;
inclua elemento em ES;
}
}caso contrário, Parar;
}

Procedimento *determina_*a_k (a_1, k)
{
Para cada dupla (q_1, q_k) em PES_k, em ordem não-decrescente de q_1\{
Se $q_1 > a_1$
pare
senão
$a_k = q_k$
}
devolva a_k
};

Note que o procedimento *determina_*a_k sempre encontra a_k, uma vez que existe uma dupla $(q_1 = b_1, q_k)$ em todo PES_k.

Apêndice 2. Prova de correção do algoritmo

Fato 1.1 Uma solução candidata $a = (a_1, \overline{a}_2, ..., \overline{a}_m)$ construída pelo passo 4 é tal que $\overline{a}_k$ é um limitante inferior para a quantidade de recursos do tipo *k*, $k = 2,...,m$, quando apenas a_1 recursos do tipo 1 estão disponíveis.

Teorema 1.2 (Invariante do *loop* do passo 5) Toda solução viável ou está em *ES* ou é dominada por algum elemento em *ES*.

Prova: No início do passo 5, o conjunto *ES* contém $|EP|$ soluções candidatas da forma $(a_1, \overline{a}_2, ..., \overline{a}_m)$, uma para cada elemento a_1 de *EP*, construídas no passo 4. Suponha, por absurdo, que existe uma solução factível $(p_1, ..., p_m)$ não pertencente a *ES* e não dominada por elementos de *ES*, no início do laço. Seja $a_1 \in EP$ tal que $a_1 = \max\{a \in EP \mid a \leq p_1\}$. Se não existir tal a_1, então $p_1 < b_1$, uma vez que $b_1 \in EP$ e, portanto, $(p_1, ..., p_m)$ é infactível. Caso contrário, existe $(a_1, ..., a_m)$ em *ES*. Uma vez que $(p_1, ..., p_m)$ não é dominado por *ES*, existe um i $(2 \leq i \leq m)$ tal que $a_i > p_i$, que é uma contradição devido a $a_1 \leq p_1$ e ao Fato 1.1.

Seja $(a_1, a_2, ..., a_m)$ uma solução infactível avaliada na iteração *j* do laço do passo 5. Uma vez que esta solução é substituída por *m* soluções candidatas aumentando-se marginalmente uma unidade de cada tipo de recurso, qualquer solução factível não pertencente a *ES* e dominada por $(a_1, a_2, ..., a_m)$ será dominada por alguma das *m* soluções candidatas incluídas em *ES*. Portanto, mantendo a invariante do laço na iteração *j*+1.

Corolário 1.3 A primeira solução factível encontrada no passo 5 é a solução ótima.

Prova: A prova segue como conseqüência direta do Teorema 1.2 e do fato de $C(a_k)$ ser uma função não-decrescente.