UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE SISTEMAS

SCATTER SEARCH PARA PROGRAMAÇÃO DE PROJETOS COM CUSTO DE DISPONIBILIDADE DE RECURSOS SOB INCERTEZA

DENISE SATO YAMASHITA

Orientador: Prof. Vinícius Amaral Armentano

Comissão Julgadora:
Celso C. Ribeiro
Cid Carvalho de Souza
Débora Pretti Ronconi
Marcos Nereu Arenales
Secundino Soares Filho
Takaaki Ohishi
Vinícius Amaral Armentano

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas – UNICAMP como parte dos requisitos exigidos para obtenção do título de DOUTOR EM ENGENHARIA ELÉTRICA.

- OUTUBRO 2003 –

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Y14s

Yamashita, Denise Sato
Scatter search para programação de projetos com custo de disponibilidade de recursos sob incerteza. / Denise Sato Yamashita. --Campinas, SP: [s.n.], 2003.

Orientador: Vinícios Amaral Armentano.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Incerteza (Teoria da informação). 2. Otimização combinatória. 3. Programação heurística. 4. Risco. I. Armentano, Vinícius Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RESUMO

Este trabalho aborda o problema de programação de projetos com o objetivo de minimizar o custo de disponibilidade de recursos, levando em consideração uma data de entrega para o projeto e relações de precedência entre as atividades. Três modelos são estudados, um modelo determinístico e dois modelos em que existe incerteza em relação à duração das atividades. Nos problemas com incerteza, as durações das atividades são representadas por um conjunto finito de possíveis cenários. Enquanto muitos estudos envolvendo incerteza concentram-se em otimizar um resultado médio, o objetivo deste trabalho é encontrar soluções que sejam também robustas, independente de qual cenário ocorra. Para resolver os três modelos, implementações do método *scatter search* são propostas. Testes computacionais foram realizados, e em problemas de médio porte as soluções geradas pelo *scatter search* são comparadas com heurísticas de múltiplos inícios. Em problemas de pequeno porte, no modelo determinístico, o desempenho do método *scatter search* foi comparado com soluções ótimas geradas por um algoritmo exato e limitantes inferiores e superiores da literatura.

Palavras-Chave: Programação de Projetos, Incerteza, Otimização Robusta, *Scatter Search*, Heurística.

ABSTRACT

This work considers the project scheduling problem with the objective of minimizing resource availability costs, taking into account a due date for the project and precedence relations among the activities. Three models are studied, a deterministic model and two models where there is an uncertainty with respect to the duration of the activities. In the problems with uncertainty, the duration of the activities are represented by a finite set of possible scenarios. While many studies involving uncertainty focus in optimizing an average result, the objective of this work is to find solutions that are also robust, independent of which scenario occurs. In order to solve the three models, we propose an approach based on the scatter search method. Computational tests were performed, and for medium size instances, the performance of the scatter search method was compared to solutions generated by multi-start heuristics. For small instances, in the deterministic model, the performance of the scatter search method was compared to optimal solutions generated by an exact algorithm and lower and upper bounds from the literature.

Keywords: Project Scheduling, Uncertainty, Robust Optimization, *Scatter Search*, Heuristics.

Para Sílvia e Sunao

AGRADECIMENTOS

A todos que colaboraram para a realização deste trabalho. Em especial

ao Prof. Vinícius Amaral Armentano, pelas valiosas discussões que muito contribuíram para este trabalho, por sua paciência e por sua dedicação;

ao Prof. Manuel Laguna, pelas sugestões que ajudaram a enriquecer este trabalho;

ao revisor da FAPESP;

à FAPESP pelo apoio financeiro;

à minha família pelo constante interesse, incentivo e carinho;

ao Sávio por estar sempre presente, por seu apoio, e por seu amor.

SUMÁRIO

INTRODUÇÃO

O gerenciamento de projetos envolve a coordenação de um grupo de atividades, no qual o gerente exerce várias funções como planejar e controlar o projeto, para atingir um determinado objetivo. Um projeto consiste de atividades que têm uma duração, relações de precedência e competem por recursos. Uma das principais áreas do gerenciamento de projetos é a programação (*scheduling*) do projeto, que consiste da alocação de recursos a atividades ao longo do tempo. As primeiras abordagens deste problema 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*) (ver Moder *et al*., 1983, para um histórico interessante). Desde então, a programação de projetos tem sido objeto de intensa pesquisa. Aplicações de programação de projetos podem ser encontradas em construções de ponte, estradas, prédios (Hartmann, 2000), manutenções de rotina em fábricas, indústria química (Cavalcante, 1998, Cavalcante *et al.*, 2001), projetos de pesquisa e desenvolvimento, como por exemplo, o projeto do míssil Polaris, nos anos 50 (Moder *et al*., 1983).

Um problema muito estudado na literatura, pela sua importância prática, é o problema de programação de projetos com recursos limitados (PPPRL), no qual procura-se minimizar o instante de término do projeto (*makespan*). Neste trabalho, estamos interessados em modelos de programação de projetos em que o tempo é escasso, isto é, existe uma data de entrega para o projeto, e os recursos são ilimitados com um custo de disponibilidade dado por uma função não-decrescente. Trata-se do problema de custo de disponibilidade de recursos (PCDR), cujo objetivo é encontrar a alocação de recursos de custo mínimo, de forma que o *makespan* não ultrapasse

uma certa data de entrega para o projeto e as relações de precedência entre as atividades sejam obedecidas.

O objetivo desta pesquisa é abordar três problemas da área de programação de projetos. O primeiro deles é o PCDR, mencionado no parágrafo anterior. O segundo e o terceiro problemas são extensões do PCDR, e consideram situações em que existe incerteza com relação à duração das atividades. A importância de estudar problemas de programação de projetos com incerteza foi reconhecida desde o desenvolvimento do método PERT, no qual se assume que a duração das atividades segue uma distribuição do tipo Beta. Pesquisas mais recentes incluem incerteza em modelos mais elaborados de programação de projetos, como o PPPRL. Apesar do progresso nesta área, ainda não existem trabalhos na literatura que abordam modelos com incerteza para o PCDR, o que estimulou um dos temas desta tese, que é o estudo do PCDR com incerteza na duração das atividades. Neste tipo de problema, o decisor deseja determinar a quantidade de recursos que deve ser alocada para o projeto, sem saber *a priori*, a duração das atividades.

Enquanto muitos estudos envolvendo incerteza otimizam um resultado médio, o objetivo deste trabalho é utilizar uma abordagem para controlar a variabilidade da solução com relação às incertezas. Controlar a variabilidade das soluções é importante em situações em que o decisor tem aversão a risco, e uma grande variabilidade significa um alto risco. Nesta abordagem, chamada de otimização robusta (Mulvey *et al*., 1995), as durações das atividades são representadas como um conjunto de possíveis cenários e procura-se uma solução que não seja muito sensível em relação a este conjunto de cenários. Dois tipos de função objetivo são propostos para controlar a variabilidade da solução, a minimização do desvio máximo (*maximum regret*), e a minimização da média-variância (*mean-variance*), que constituem o segundo e terceiro problemas aqui estudados. É importante ressaltar que a solução, tanto do problema de desvio máximo, como do problema de média-variância, envolvem um equilíbrio entre minimizar o custo do projeto e minimizar o risco.

Devido à complexidade computacional dos três problemas abordados, métodos ótimos de resolução são inviáveis, o que nos remete à utilização de métodos heurísticos. Neste trabalho, implementações do método populacional *scatter search* são desenvolvidas e aplicadas aos três modelos do PCDR apresentados acima. As idéias originais do método *scatter search* foram

propostas por Glover (1977) e um modelo (*template*) detalhado do método foi apresentado em Glover (1998). Desde então, estratégias mais avançadas e implementações alternativas têm sido propostas na literatura (Glover *et al*. (2002), Laguna e Martí (2002), Laguna e Armentano (2002)). Métodos populacionais geram um conjunto de soluções a cada iteração através de operações específicas aplicadas ao conjunto de soluções anterior. A função dos operadores é propiciar diversificação das soluções e intensificação da busca em certas regiões do espaço de soluções. Um método populacional largamente utilizado é o algoritmo genético que, em geral, utiliza operações probabilísticas de reprodução, *crossover*, mutação e possivelmente uma operação de busca local na população de soluções. O método *scatter search* pode ser visto como um algoritmo evolutivo ou populacional, que constrói soluções a partir da combinação de outras, e é baseado em estratégias determinísticas bem definidas de exploração de subconjuntos das populações.

Esta dissertação está organizada da seguinte forma. O Capítulo 1 descreve o PCDR e propõe um método de resolução, baseado na heurística *scatter search*. Diversos métodos de combinação de solução e estratégias avançadas para atualizar o conjunto de referência são propostos. Este capítulo também descreve a implementação de heurísticas simples, de múltiplos inícios, que são utilizadas para avaliar o desempenho do *scatter search*. O Capítulo 2 descreve como os problemas testes foram gerados e relata os resultados computacionais, mostrando a eficácia do método proposto. Para problemas de pequeno porte, as soluções geradas pelo *scatter search* são comparadas com soluções exatas, limitantes inferiores e superiores da literatura. Para problemas de médio porte, são utilizadas heurísticas simples, de múltiplos inícios, com a finalidade de mostrar o ganho de qualidade gerado pelo *scatter search*, que tem estratégias mais complexas do que a heurística de múltiplos inícios.

O Capítulo 3 apresenta uma extensão do problema de programação de projetos PCDR, considerando incerteza nas durações das atividades. Dois modelos de otimização robusta para o PCDR são abordados, o modelo de minimização do desvio máximo e o modelo de minimização da média-variância. O capítulo descreve também como o método *scatter search* e a heurística de múltiplos inícios são adaptados para resolver os dois modelos. Os resultados computacionais são relatados no Capítulo 4 e mostram que o método *scatter search* é adequado para resolver os

problemas com cenários. As diferenças entre as estratégias de minimização do desvio máximo e minimização de média-variância também são discutidas. Finalmente, no Capítulo 5, apresentam-se resumidamente, as principais conclusões deste trabalho, assim como sugestões de tópicos de pesquisa que podem ser desenvolvidos no futuro.

CAPÍTULO 1

O PROBLEMA DE CUSTO DE DISPONIBILIDADE DE RECURSOS

Este capítulo apresenta o problema de custo de disponibilidade de recursos, e descreve os procedimentos de solução baseados no método *scatter search*. São apresentadas também heurísticas de múltiplos inícios, utilizadas neste trabalho para mostrar o ganho de qualidade gerado pelo *scatter search*.

1.1 Descrição do problema

Um projeto determinístico consiste de atividades que têm uma duração, relações de precedência e competem por recursos. Problemas de programação de projetos consistem em encontrar um instante de início para todas as atividades, e um objetivo comum na prática é minimizar o instante de término do projeto (*makespan*) quando os recursos são escassos. Tal problema, conhecido como problema de programação de projetos com recursos limitados (PPPRL), tem sido estudado extensivamente na literatura. Blazewicz *et al.* (1983) mostram que este problema é *NP-hard*, e revisões de literatura recentes sobre métodos exatos e heurísticos podem ser encontradas em Herroelen *et al*. (1998), Brucker *et al*. (1999), Kolisch e Hartmann (1998), e Hartmann e Kolisch (2000).

Neste capítulo, o problema de custo de disponibilidade de recursos (PCDR) é abordado. O PCDR distingue-se do PPPRL no sentido em que o tempo é escasso e os recursos, ilimitados,

têm um custo de disponibilidade dado por uma função não-decrescente. O objetivo do PCDR é encontrar a alocação de recursos de custo mínimo, de forma que o *makespan* não ultrapasse uma data de entrega para o projeto. Este problema foi proposto por Möhring (1984), motivado por um projeto de construção de ponte, de pequeno porte. Experimentos computacionais são relatados e o autor também mostra que o problema é *NP-hard*. Nos problemas reais de gerenciamento de projetos, sempre existe a questão do equilíbrio entre tempo e custo (Krajewski e Ritzman, 1996). Por exemplo, um projeto em geral pode ser completado mais cedo se existe uma disponibilidade maior de recursos. Entretanto, quanto mais recursos forem disponibilizados, maior será o custo do projeto. Por outro lado, alocar poucos recursos pode resultar em atraso no projeto. Drexl e Kimms (2001) destacam que a solução do PCDR para diversas datas de entrega, fornece o equilíbrio de tempo/custo, que podem ser uma informação valiosa quando deseja-se negociar um preço para um projeto. Uma descrição mais detalhada do PCDR é dada a seguir.

Considere um projeto com n atividades em que as atividades 1 e n são atividades artificiais que indicam o início e fim do projeto, respectivamente. O projeto é sujeito a relações de precedência entre as atividades, e H é o conjunto dos pares de atividades (i, j), tal que $(i, j) \in H$ se a atividade i precede a atividade j no projeto. Cada atividade i tem duração d_i e requer r_{ik} unidades do recurso do tipo k ($k = 1, ..., m$) durante o seu processamento. Seja A_t o conjunto das atividades em progresso durante o intervalo de tempo $(t\text{-}1,t]$ e D, a data de entrega do projeto. Seja $C_k(a_k)$ uma função de custo não decrescente associada à disponibilidade a_k do recurso do tipo k. As variáveis do problema são as variáveis inteiras de disponibilidade de recurso a_k e os instantes de término f_i das atividades. Um modelo conceitual para o PCDR é dado a seguir.

Minimizar $\sum_k C_k(a_k)$

sujeito a

$$f_j \geq f_i + d_j \qquad \forall (i, j) \in H$$

$$f_1 = 0$$

$$f_n \leq D$$

$$\sum_{i \in A_t} r_{ik} \leq a_k \qquad k = 1,...,m \quad \text{e} \quad t = 1,....,f_n$$

A função objetivo do modelo consiste da minimização da soma dos custos de disponibilidade dos recursos. A primeira restrição refere-se às relações de precedência entre as atividades, a segunda e a terceira restrições indicam, respectivamente, que o projeto deve começar no instante zero e terminar antes da data de entrega. A última restrição do modelo indica que a quantidade do recurso do tipo k requisitada pelas atividades em progresso no intervalo $(t-1,t]$ é limitada pela disponibilidade deste recurso. Na literatura, os testes computacionais dos trabalhos que abordam o PCDR utilizam uma função de custo $C_k(a_k)$ linear. Da mesma forma, neste trabalho, os testes computacionais são gerados utilizando uma função linear para o custo, apesar do método de resolução desenvolvido ser capaz de lidar tanto com funções de custo lineares como não-lineares.

A literatura em métodos de solução para o PCDR não é vasta. Möhring (1984) propõe um método exato de solução baseado num procedimento desenvolvido por Radermacher (1985) para o PPPRL. O autor considera, para toda disponibilidade de recursos e custos, relações de precedência que estendem as relações de precedência originais do problema e respeitam a data de entrega do projeto. A construção destas relações de precedência e a determinação da solução ótima são baseadas em teoria de grafos. Demeulemeester (1995) propõe um procedimento exato de planos de corte que funciona da seguinte forma. Para uma dada disponibilidade de recursos, o *makespan* mínimo é obtido utilizando um algoritmo *branch-and-bound* para o PPPRL (Demeulemeester, 1992). Se o *makespan* exceder a data de entrega, então a disponibilidade de um tipo de recurso é acrescida de uma unidade. O procedimento termina quando o *makespan* é menor ou igual à data de entrega. Testes computacionais no problema de construção de ponte indicam que o algoritmo de planos de corte é mais rápido do que o método proposto por Möhring (1984). Rangaswamy (1998) propõe um *branch-and-bound* para o PCDR e o aplica no mesmo conjunto de problemas testes utilizado por Demeulemeester (1995). Resultados computacionais não são conclusivos devido a diferenças nas plataformas computacionais utilizadas. Drexl e Kimms (2001) propõem procedimentos que geram limitantes inferiores para o PCDR, baseados em relaxação lagrangiana e método de geração de colunas. Soluções factíveis são disponibilizadas pelos dois métodos. Gagnon *et al*. (2002) abordam uma variante do PCDR, em que a função objetivo $\sum_k C_k(a_k)$, é multiplicada pelo *makespan*. O procedimento de solução proposto pelos autores pode ser adaptado para resolver o PCDR, e utiliza regras de despacho para

obter uma solução de partida factível, que corresponde ao programa das atividades e uma disponibilidade de recursos. Uma busca tabu, baseada em movimentos de inserção de atividades no programa, é aplicada com o objetivo de minimizar o *makespan*, isto é, a busca tabu resolve um PPPRL. Se a busca tabu conseguir encontrar um programa com menor valor de *makespan* para a disponibilidade de recursos dada, então, o procedimento verifica se todos os recursos disponíveis são utilizados em sua totalidade. Caso existam recursos com folga, então, esta disponibilidade de recursos é reduzida até que a folga desapareça. Seu enfoque, portanto, é minimizar o *makespan* e obter a disponibilidade de recursos necessária. Experimentos computacionais são realizados em 110 problemas testes, com até 51 atividades. De nosso conhecimento, esta é a única heurística relacionada ao PCDR proposta na literatura.

Considere um exemplo do PCDR com 12 atividades, $m = 3$ tipos de recursos e data de entrega $D = 18$. A Figura 1.1 mostra as relações de precedência entre as atividades e a Tabela 1.1 mostra a duração das atividades e a quantidade de recursos que cada atividade requer para ser processada.

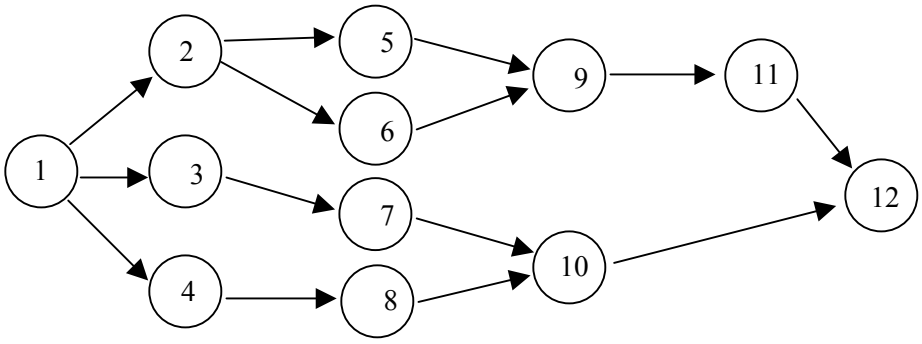

Figura 1.1 Relações de precedência

Tabela 1.1 Dados para o projeto do Exemplo 1.

	Atividades											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
r_{i1}	0	0	2	0	2	0	1	0	2	2	0	0
r_{i2}	0	0	0	1	0	1	1	0	2	0	0	0
r_{i3}	0	1	1	2	0	0	0	1	1	0	2	0
d_i	0	2	3	4	2	5	2	5	4	3	7	0

A Figura 1.2 mostra a programação das atividades para um dado vetor de disponibilidade de recursos, a = (2,2,2). O projeto termina no instante 25, isto é, ele é infactível com respeito à data de entrega.

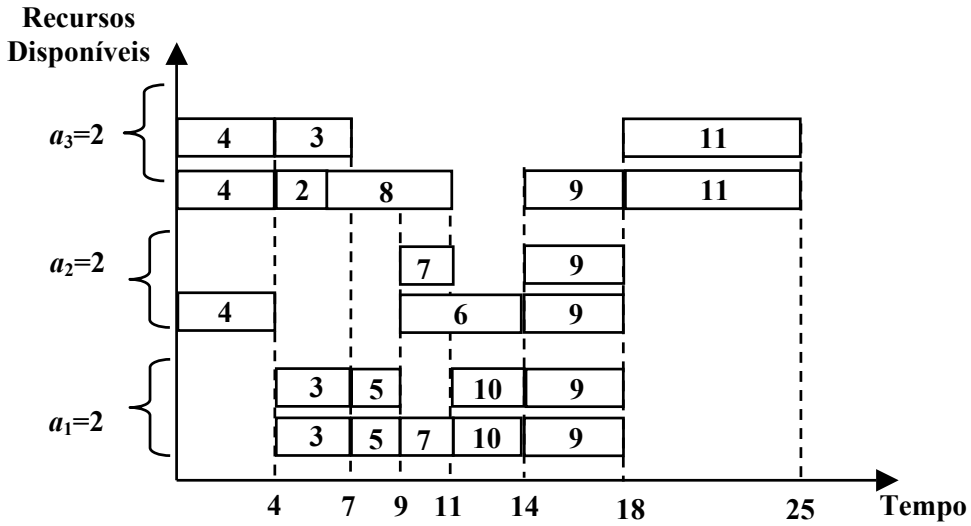

Figura 1.2 Programação das atividades para o Exemplo 1.

1.2 Heurística de melhoria para o PCDR

Esta Seção descreve uma heurística de melhoria que é utilizada no método *scatter search* e na heurística de múltiplos inícios. Esta heurística é baseada em movimentos realizados nas variáveis de disponibilidade de recurso. Note que quando estas variáveis assumem um valor particular, o *makespan* mínimo deve ser calculado para verificar a factibilidade relativa à data de entrega do projeto. Neste sentido, as variáveis de disponibilidade de recurso são as variáveis de decisão ou variáveis de controle, e portanto, uma *solução* consiste de valores designados para tais variáveis. Encontrar o *makespan* mínimo é equivalente a resolver o PPPRL e a utilização de um método de solução exato tornaria a heurística inviável. Portanto, é necessário aplicar um procedimento de solução para o PPPRL que ofereça um bom equilíbrio entre qualidade de solução e tempo computacional. O procedimento de busca adaptativo (*ASP – adaptive search procedure*), proposto por Kolisch e Drexl (1996), possui estas características, e é utilizado neste trabalho. Este procedimento é um método *multi-pass* que combina regras de prioridade e um

esquema de geração de vários programas (*schedules*). Kolisch e Hartmann (1998) fazem uma revisão de tais métodos e apresentam comparações. Uma descrição do ASP pode ser encontrada no Apêndice A.

A heurística de melhoria é composta de duas fases: viabilização (Fase 1) e melhoria (Fase 2). Ao longo deste trabalho, consideraremos uma solução infactível se o *makespan* do programa gerado pela heurística ASP exceder a data de entrega. A Fase 1 é aplicada quando a solução corrente é infactível, e neste caso, a quantidade de recursos disponíveis deve ser aumentada. Tal aumento é baseado no instante de término (f_i) das atividades no programa infactível, e no instante de término mais tarde (LF_i) das atividades. Para o cálculo de LF_i assume-se um valor para o *makespan* igual à data de entrega; este cálculo pode ser realizado *a priori*, pois não depende da disponibilidade de recursos. O próximo passo é selecionar a atividade p com a maior diferença entre f_i e LF_i, isto é, a atividade mais atrasada. A disponibilidade de recursos da solução corrente $a = (a_1,...,a_m)$, é então aumentada pelo requisito de recursos r_{pk}, $k = 1,..., m$, resultando em $a' = (a_1 + r_{p1}, ..., a_m + r_{pm})$ e ASP é aplicada para gerar um novo programa. Se a solução continuar infactível, o aumento de recursos é descartado e o processo é repetido a partir da solução corrente original $a = (a_1,...,a_m)$, utilizando desta vez, a quantidade de recursos requerida pela atividade com o segundo maior valor de $(f_i - LF_i)$. Se a nova solução também for infactível, o procedimento pára. Caso contrário, a Fase 2 é iniciada. O número de tentativas para obter uma solução factível é restrita a duas, com a finalidade de limitar o tempo computacional gasto pela heurística de melhoria, que é chamada diversas vezes pelos procedimentos heurísticos *scatter search* e método de múltiplos inícios.

A Fase 2 consiste em decrescer a quantidade de recursos disponíveis, um tipo por vez, de uma unidade. Se uma nova solução factível com um custo menor for encontrada, a solução corrente é atualizada. Este procedimento é repetido até que não seja mais possível decrescer a disponibilidade de recursos sem gerar uma solução infactível.

Sempre que uma solução é gerada na Fase 1 e Fase 2, uma tentativa é feita para eliminar o folga de disponibilidade de recursos. Suponha que a disponibilidade de recurso k, seja a_k, e que a quantidade de recurso do tipo k utilizada ao longo do projeto seja $b_k < a_k$. Então, a disponibilidade de recurso k é reduzida a b_k. Este procedimento é denominado procedimento de eliminação de

folgas (PEF). O pseudocódigo da heurística de melhoria é mostrado na Figura 1.3 e um exemplo da aplicação da heurística é apresentado abaixo.

Considere o projeto descrito no exemplo da Seção 1.1, e a solução corrente infactível dada por a = (2,2,2). A Tabela 1.2 mostra o instante de término (f_i), o instante de término mais tarde (LF_i), e a diferença entre o instante de término e o instante de término mais tarde ($f_i - LF_i$) para cada atividade. A Fase 1 da heurística de melhoria é aplicada. Uma vez que as atividades 6, 9 e 11 apresentam o maior valor de ($f_i - LF_i$), a atividade 9 é escolhida arbitrariamente. A Tabela 1.1 mostra que a quantidade de recursos necessários para que a atividade 9 seja processada é $r_{9,1}$=2, $r_{9,2}$=2, $r_{9,3}$=1. Portanto, a disponibilidade de recursos é aumentada para a'=(2+2,2+2,2+1)=(4,4,3). A aplicação de ASP nesta solução, resulta num programa com instantes de término mostrados na Tabela 1.2, e ilustrado na Figura 1.4. A seguir, a folga do recurso do tipo 2 é eliminada por PEF, gerando uma nova solução $a = a'$ = (4,2,3). Esta solução é factível, e a Fase 2 é iniciada.

Iteração 1: solução corrente: a = (4,2,3)

a' = (4-1,2,3) = (3,2,3) – ASP gera um programa factível com instantes de término exibidos na última linha da Tabela 1.2. A aplicação do PEF resulta em a' = (2,2,3), que tem um custo menor, e portanto, a = (2,2,3).

Iteração 2: solução corrente: a = (2,2,3)
a' = (2-1,2,3) = (1,2,3) – recurso insuficiente
a' = (2,2-1,3) = (2,1,3) – recurso insuficiente
a' = (2,2,3-1) = (2,2,2) – programa infactível
Fim da Fase 2.

procedimento Heurística_de_Melhoria (a)

Passo 1: Se a solução for infactível, vá para o **Passo 2**, caso contrário, vá para o **Passo 3**.

Passo 2: Fase 1.

Passo 2.1. Primeira tentativa: Calcule $(f_i - LF_i)$ e selecione a atividade p com valor máximo $(f_i - LF_i)$. Aumente a quantidade de recursos da solução corrente $a = (a_1,...,a_m)$, na quantidade de recursos requeridos pela atividade p: $a' = (a_1 + r_{p1}, ..., a_m + r_{pm})$. Aplique ASP e PEF. Se a solução for factível, vá para o **Passo 3**, caso contrário, vá para o **Passo 2.2.**

Passo 2.2. Segunda tentativa: Calcule $(f_i - LF_i)$ e selecione a atividade q com o segundo maior valor de $(f_i - LF_i)$. Aumente os recursos da solução corrente: $a' = (a_1 + r_{q1}, ..., a_m + r_{qm})$. Aplique ASP e PEF. Se a solução for factível, vá para o **Passo 3**, caso contrário, pare.

Passo 3: Fase 2.

Para cada recurso $k = 1,.., m$, faça:

Reduza o recurso de uma unidade: $a'_k = a_k - 1$. Aplique ASP e PEF. Se a solução for factível, atualize a solução corrente: $a_k = a'_k$ e vá para o **Passo 4**.

Passo 4: Se houve melhoria na solução no **Passo 3**, repita o **Passo 3**, caso contrário, pare e **retorne**(a).

Figura 1.3 Pseudocódigo da heurística de melhoria

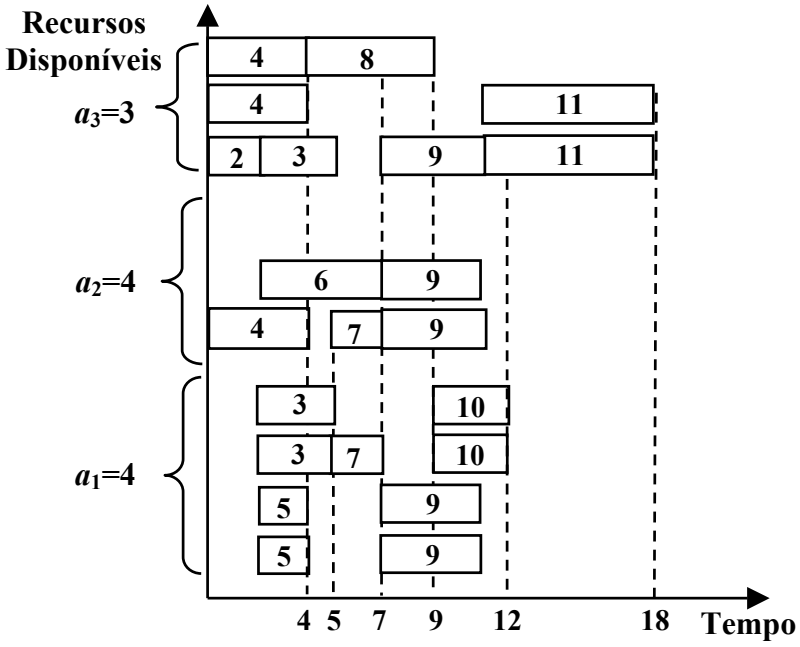

Figura 1.4 Programação das atividades – Fase 1, antes de aplicar PEF

Tabela 1.2 Aplicando a heurística de melhoria no exemplo.

	Atividades											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
f_i	0	6	7	4	9	14	11	11	18	14	25	25
LF_i	0	2	13	10	7	7	15	15	11	18	18	18
$f_i - LF_i$	-	4	-6	-6	2	7	-4	-4	7	-4	7	-
f_i – fim da Fase1	0	2	5	4	4	7	7	9	11	12	18	18
f_i - Iteração 1 da Fase 2.	0	2	5	4	7	7	13	9	11	16	18	18

1.3 *Scatter search*

O método *scatter search* (SS) é um método populacional que tem mostrado grande potencial para resolver problemas de otimização combinatória e não-linear. As idéias originais do método foram propostas por Glover (1977) e um modelo (*template*) detalhado do método foi apresentado em Glover (1998). Métodos populacionais geram um conjunto de soluções a cada iteração através de operações específicas aplicadas ao conjunto de soluções anterior. A função dos operadores é propiciar diversificação das soluções e intensificação da busca em certas regiões do espaço de soluções. Um método populacional largamente utilizado é o algoritmo genético que, em geral, utiliza operações probabilísticas de reprodução, *crossover*, mutação e possivelmente uma operação de busca local na população de soluções. SS difere de algoritmos genéticos, pois utiliza estratégias determinísticas para atingir diversificação e intensificação. SS contém um gerador sistemático de soluções diversas, que leva em consideração as características do espaço de soluções. Além disso, o método explora o espaço de solução sistematicamente, em relação a um conjunto de pontos de referência, que tipicamente consiste de boas soluções obtidas ao longo da busca. Observe que o critério para que uma solução seja considerada "boa" não fica restrito aos valores da função objetivo, mas também leva em consideração sua diversidade em relação às outras soluções do conjunto de referência.

Veja Laguna e Martí (2003) para implementações mais avançadas, incluindo uso de memória e *path relinking*, e aplicações em diversos problemas de otimização combinatória e não-linear. Outros trabalhos que discutem implementações de *scatter search* podem ser encontrados em Glover *et al*. (2000), Laguna e Armentano (2002), Glover *et al*. (2002), Laguna (2002).

O enfoque do método *scatter search* pode ser descrito abaixo de forma sucinta pelo seguinte modelo (*template*):

1) Método de Geração de Soluções Diversas: utilizado para gerar um conjunto de soluções tentativas diversas, isto é soluções distantes entre si.

2) Método de Melhoria: procedimento heurístico, em geral, a ser aplicado nas soluções tentativas.

3) Método para atualização do conjunto de referência: é utilizado para atualizar o conjunto de referência; este conjunto é pequeno, tipicamente contém 20 soluções e é dividido em um subconjunto de soluções de alta qualidade e um subconjunto de soluções diversas.

4) Método de geração de subconjuntos: é utilizado para gerar subconjuntos de soluções do conjunto de referência; as soluções de cada subconjunto são combinadas.

5) Método para gerar combinação de soluções: uma combinação de soluções procura extrair informação relevante de elementos de soluções selecionadas e combiná-los através de um procedimento; o método de melhoria é aplicado às melhores soluções geradas pelas combinações.

Os principais parâmetros do *scatter search* são descritos a seguir:

PSize = tamanho da população inicial *P*

b = tamanho do conjunto de referência (*RefSet*)

b_1 = tamanho do subconjunto de alta qualidade

b_2 = tamanho do subconjunto diverso

MaxIter = número máximo de iterações

MaxSol = número máximo de soluções avaliadas

λ e *g* = parâmetros utilizados no método de geração de soluções diversas

Um esboço do método *scatter search* é apresentado na Figura 1.5. O primeiro passo constrói um conjunto inicial de soluções *P*, com *PSize* elementos. Estas soluções são submetidas ao método de melhoria e são denominadas *solucões melhoradas*. Os elementos de *P* são obtidos pelo método de geração de soluções diversas, baseado em memória de freqüência, detalhado na Seção 1.3.1. O Passo 2 seleciona *b* soluções de *P*, de acordo com sua qualidade ou diversidade, com a finalidade de construir o conjunto de referência (*RefSet*), que é uma coleção, tanto de soluções de alta qualidade, como também de soluções diversas. A construção e atualização de *RefSet* é discutida na Seção 1.3.2. No Passo 3, as soluções de *RefSet* são agrupadas em subconjuntos de soluções descritos na Seção 1.3.3, e as soluções de cada subconjunto são

combinadas. Os diversos tipos de combinação estudados neste trabalho são descritos na Seção 1.3.4. Após gerar uma solução combinada, a heurística de melhoria é aplicada a esta solução. Neste ponto, o conjunto de referência pode ser atualizado dependendo do critério de atualização do conjunto de referência, que é descrito mais adiante, na Seção 1.3.2. Se a busca convergir, isto é, nenhuma solução nova é elegível para entrar em *RefSet*, então o Passo 4 reconstrói *RefSet*. Isto corresponde a uma iteração do *scatter search*. O algoritmo pára depois de *MaxSol* soluções avaliadas ou *MaxIter* iterações realizadas, dependendo da versão do *scatter search*.

Procedimento scatter_search( )

Passo 0. $P = \varnothing$

Passo 1. Gerar o conjunto *P* de soluções diversas: $P \leftarrow$Gera_sol_diversas ($\lambda, g, PSize$).

Passo 2. Construir o conjunto de referência – Selecione as b_1 melhores soluções e as b_2 soluções mais diversas de P para construir o conjunto de referência (*RefSet*). *Número de iterações* = 0.

enquanto (número de soluções avaliadas < *MaxSol*) ou (*Número de iterações* < *MaxIter*)

{

Passo 3. Realizar combinação e atualizar *RefSet*

Passo 4. Reconstruir *RefSet* – Remova as b_2 piores soluções de *RefSet*. Gere *PSize* soluções melhoradas: $P \leftarrow$Gera_sol_diversas ($\lambda, g, PSize$). Escolha as b_2 soluções diversas e adicione-as a *RefSet*. *Número de iterações* = *Número de iterações* + 1.

}

Passo 5. Retorne melhor solução encontrada.

Figura 1.5 Método *scatter search*

1.3.1 Método de geração de soluções diversas

A coleção de soluções tentativas diversas P é gerada através de informação obtida ao longo da busca. Seja $[z_k^g, z_k^{g+1})$ um intervalo de inteiros contendo os possíveis valores de disponibilidade de recurso para o recurso do tipo k. Suponha que este intervalo é dividido em g

subintervalos $[z_k^0, z_k^1)$, …, $[z_k^{i-1}, z_k^i)$,…, $[z_k^{g-1}, z_k^g)$, com a mesma cardinalidade. Defina uma matriz de freqüência $M[k][i]$, $k = 1,...,m$, , $i = 0,..., g\text{-}1$ que armazena o número de vezes que a variável a_k assume um valor no subintervalo $[z_k^i, z_k^{i+1})$. Um subintervalo é então selecionado com probabilidade inversamente proporcional à sua freqüência em M, e um valor para a_k é gerado aleatoriamente em tal intervalo. A Figura 1.6 mostra o método de geração de soluções diversas. O Passo 0 só é utilizado para gerar as soluções da população inicial, e consiste em determinar os valores dos extremos do intervalo associado ao recurso k. O limitante inferior z_k^0 recebe o valor l_k,

$$l_k = \max_{i=1,...,n} \{r_{ik}\}, \qquad k = 1,..., m$$

que é um limitante inferior para a disponibilidade do recurso k. Um possível valor para z_k^g é o limitante superior u_k para a disponibilidade do recurso k,

$$u_k = \sum_{i=1,...,n} \{r_{ik}\}, \qquad k = 1,..., m.$$

Entretanto, este limitante superior é muito alto, e um valor mais razoável para z_k^g é definido a seguir. Inicialmente, a solução é gerada por uma distribuição uniforme U(l_k , u_k). A heurística de melhoria descrita na Seção 1.2 é aplicada e, se uma solução factível $H = (h_1,...,h_m)$ for obtida, então, para cada tipo de recurso k, $z_k^g = \lambda \cdot h_k$, tal que $\lambda \geq 1$ é um parâmetro de entrada. Se após aplicar a heurística de melhoria, a solução resultante for infactível, então o Passo 0 é repetido até que uma solução factível seja encontrada. Durante a busca, existe a possibilidade da disponibilidade de recursos a_k assumir um valor tal que, $a_k > z_k^g$. Neste caso, z_k^g é atualizado, $z_k^g = \lambda \cdot a_k$, e os subintervalos são calculados novamente. No próximo passo do procedimento de geração de soluções diversas, Passo 1, a matriz de freqüência é utilizada para gerar uma solução diversa. A heurística de melhoria é aplicada à solução corrente no Passo 2 e a matriz de freqüência é atualizada. Se a solução gerada no Passo 2 for infactível, então ela é incluída em P com um custo alto. O procedimento é repetido até que *PSize* soluções sejam geradas.

procedimento Gera_ sol_diversas ($\lambda, g, PSize$ **)**

Passo 0.

Se geração da população inicial faça:

{

Calcule z_k^g : gere uma solução *a* através de uma distribuição uniforme U(l_k , u_k).

H←Heurística_de_Melhoria (*a*), e obtenha uma solução $H = (h_1,...,h_m)$.

Se solução *H* for factível, então

{

Faça $z_k^g = \lambda \cdot h_k$, k = 1, ..., m. Inicialize $M[k][i] = 1$, k = 1,..., m, i = 0,.., g-1. Vá para o **Passo 1**.

}

caso contrário, repita o **Passo 0**.

}

caso contrário, vá para o **Passo 1**.

Passo 1. Para cada recurso k = 1,..., m, gere valores para a_k : primeiro, selecione um subintervalo com probabilidade inversamente proporcional a $M[k][i]$, e então, gere aleatoriamente um valor para a_k , dentro do subintervalo selecionado.

Passo 2. Aplique *a*←Heurística_de_Melhoria (*a*). Inclua a solução melhorada em *P*.

Passo 2.1. Atualize a matriz de freqüência *M*.

Passo 3. Repita os **Passos 1** e **2**, até que *PSize* soluções sejam geradas. **retorne**(*P*).

Figura 1.6 Procedimento para gerar soluções diversas

1.3.2 Método para construir e atualizar o conjunto de referência

Este método é utilizado para criar e manter um conjunto de soluções de referência. Soluções são incluídas no conjunto de referência de acordo com sua qualidade ou diversidade. O conjunto de referência pode ser dividido em dois subconjuntos: subconjunto de soluções de qualidade (*RefSet*1), que contém as b_1 melhores soluções, em termos de valor de função objetivo, e o subconjunto de soluções diversas (*RefSet*2), que consiste de b_2 soluções diversas.

Como criar o conjunto de referência

Considere o conjunto P de soluções diversas melhoradas geradas no Passo 1 da Figura 1.5. *RefSet*1 é composta pelas b_1 melhores soluções pertencentes a P. *RefSet*2 é construída a partir dos elementos de P que não pertencem a $RefSet = RefSet1 \cup RefSet2$, e que maximizam a distância mínima a *RefSet*. O primeiro elemento de *RefSet*2 é obtido da seguinte forma: para cada elemento de P não pertencente a *RefSet*, calcula-se a distância deste elemento aos demais elementos de *RefSet*, (observe que neste estágio, *RefSet* só contém elementos de *RefSet*1), e dentre os valores de distância calculados, avalia-se a distância mínima. A distância entre dois elementos x e y é medida por,

$$d(x, y) = \sum_{k=1,\ldots,m} |x_k - y_k|$$

A distância mínima de um elemento x em relação aos elementos de *RefSet* é dada por,

$$dmin(x) = \min_{y \in RefSet} d(x, y)$$

Após calcular o valor de $dmin(x)$ para todos os elementos $x \in P$ não pertecentes a *RefSet*, seleciona-se o elemento x com maior valor de distância mínima. Este é o elemento que maximiza a distância mínima, e portanto, é inserido em *RefSet*2, e conseqüentemente, em *RefSet*. O

próximo elemento de *RefSet*2 é obtido de forma semelhante: para cada elemento de *P* não pertencente a *RefSet*, calcula-se a distância mínima deste elemento aos demais elementos de *RefSet*, e seleciona-se o elemento de *P* que maximiza a distância mínima. Este procedimento é repetido até que *RefSet*2 tenha b_2 elementos.

Atualização do conjunto de referência

Existem dois aspectos que devem ser considerados quando *RefSet* é atualizado. O primeiro refere-se ao momento em que a atualização deve ser realizada. Existem duas possibilidades:

Atualização estática (S). Na atualização estática, o conjunto de referência não muda até que todas as combinações de soluções de *RefSet* tenham sido realizadas. Exemplos de tais tipos de atualização podem ser encontradas em Campos *et al.* (2001), Laguna e Martí (2002), Campos *et al*. (2003), Martí *et al*. (2002).

Atualização dinâmica (D). Na atualização dinâmica, o conjunto de referência é atualizado sempre que uma nova solução, gerada por uma combinação, qualifica-se para entrar em *RefSet*. Neste caso, algumas combinações do *RefSet* original não são realizadas. Em geral, este tipo de atualização acelera a convergência do método *scatter search*, porque soluções de alta qualidade rapidamente substituem soluções de baixa qualidade. Exemplos de usos da atualização dinâmica são encontrados em Martí *et al*. (2000), Hamiez e Hao (2002), Greistorfer (2003), Jain e Meeran (2002), Laguna e Martí (2002), Valls *et al*. (1998).

O segundo aspecto a ser considerado na atualização de *RefSet* refere-se aos critérios de aceitação ou eliminação de uma solução de *RefSet*. Dois tipos de atualização são estudados: por qualidade e diversidade (QD) e por qualidade apenas (Q).

Atualização QD: Considere uma solução gerada pelo método de combinação de soluções. Se o custo da solução resultante da combinação for menor do que o pior elemento de *RefSet*1, então

esta solução substitui o pior elemento de *RefSet*1. Caso contrário, esta solução é inserida em *RefSet*2 se aumentar a menor distância entre as soluções deste subconjunto. Neste caso, a solução pertencente a *RefSet*2 que possuir a menor distância, em relação às demais soluções, é substituída pela solução que entra.

Atualização Q: Se a nova solução gerada pelo método de combinação de soluções tiver melhor qualidade que o pior elemento de *RefSet*, então substitui-se o pior elemento de *RefSet* pela nova solução.

O pseudocódigo da atualização estática é ilustrado na Figura 1.7. Inicialmente, o método de geração de subconjuntos é utilizado (Passo 3.2) para criar os subconjuntos de soluções a serem combinadas. Estes subconjuntos são guardados em *NewSubsets*. Note que, para um subconjunto de soluções ser incluído em *NewSubsets,* ao menos um elemento de $s \in NewSubsets$ deve ser novo. No Passo 3.4, um dos métodos de combinação de solução é aplicado a um subconjunto de soluções *s*, gerando um solução tentativa *x*. A seguir, a heurística de melhoria é aplicada a *x*. O Passo 3.5 atualiza *RefSet*, seja por qualidade e diversidade ou por qualidade somente. Note que $RefSet = RefSet1 \cup RefSet2$ e *RefSet* é ordenado por qualidade, e a solução com o melhor valor de função objetivo é a primeira da lista. O Passo 3.9 elimina o subconjunto *s* de *NewSubsets.* Quando *NewSubsets* estiver vazio, o procedimento de geração de subconjuntos é repetido, se o conjunto de referência possuir uma nova solução, isto é, *NewSolutions* = TRUE, e se o número máximo de soluções avaliadas não tiver sido atingido.

Passo 3. Combinação com atualização estática

Passo 3.1 Faça *NewSolutions* = TRUE

enquanto (*NewSolutions*) e (número de soluções avaliadas < *MaxSol*) **faça {**

Passo 3.2. Geração de subconjunto: gere *NewSubsets*, que consiste de subconjuntos de soluções geradas pelo método de geração de subconjuntos. Faça *NewSolutions* = FALSE.

enquanto (*NewSubsets* ≠ ∅) e (número de soluções avaliadas < *MaxSol*) **faça {**

Passo 3.3. Selecione o próximo subconjunto *s* de *NewSubsets*

Passo 3.4. Aplique o método de combinação de solução em *s* para obter uma nova solução tentativa *x*. *x*←Heurística_de_Melhoria (*x*).

Passo 3.5. Atualize o conjunto de referência

se atualização por qualidade e diversidade, **então {**

se *x* tem qualidade melhor do um elemento de *RefSet*1 **então {**

Passo 3.6. Inclua *x* em *RefSet*1, e elimine o pior elemento, em termos de valor de função objetivo, *RefSet*1. Faça *NewSolutions* = TRUE.

} caso contrário se *x* aumenta a distância mínima dos elementos em *RefSet*2 **então {**

Passo 3.7. Inclua *x* em *RefSet*2, e elimine o elemento com o menor valor de distância. Faça *NewSolutions* = TRUE

}

}caso contrário, se atualização por qualidade somente, **então {**

se *x* tem qualidade melhor do que o pior elemento de *RefSet,* em termos de valor de função objetivo, **então {**

Passo 3.8. Inclua *x* em *RefSet*, e elimine o pior elemento em termos de valor de função objetivo. Faça *NewSolutions* = TRUE.

}

}

Passo 3.9. Elimine *s* de *NewSubsets.*

}

}

Figura 1.7 Combinação com atualização estática

A atualização dinâmica ilustrada na Figura 1.8 começa gerando os subconjuntos de soluções, em ordem lexicográfica, com o método de geração de subconjuntos (Passo 3.1). Estes subconjuntos são guardados em *NewSubsets*. No Passo 3.2, o próximo subconjunto *s* pertencente a *NewSubsets*, é selecionado. No Passo 3.3, o método de combinação é aplicado ao subconjunto de soluções *s*, resultando em uma nova solução tentativa *x*. Em seguida, a heurística de melhoria é aplicada a *x*. Se *x* respeita os critérios para ser incluída em *RefSet* (Passos 3.4-3.6), *RefSet* e *NewSubsets* precisam ser reconstruídos. Suponha que o *RefSet* inicial é dado por *RefSet* = $\{y^1, y^2, y^3, y^4\}$, e que a combinação de (y^1, y^2) resulta numa solução *x*, em que *x* tem um valor de função objetivo menor que y^3 e maior que y^2. Note que *RefSet* está ordenado por valor de função objetivo, no qual a solução de melhor valor de função objetivo ocupa a primeira posição da lista. Portanto, o novo *RefSet* é dado por *RefSet* = $\{y^1, y^2, x, y^3\}$. O Passo 3.7 constrói dois subconjuntos de soluções, *NewSubsets*1 e *NewSubsets*2, no qual, *NewSubsets* = *NewSubsets*1 $\cup$ *NewSubsets*2. *NewSubsets*1 consiste de subconjuntos de soluções geradas pelo método de geração de subconjuntos, no qual para todos os subconjuntos *s* em *NewSubsets*1, *x* é um elemento de *s*, e *NewSubsets*2 consiste de subconjuntos de soluções geradas pelo método de geração de subconjuntos, no qual para todos os subconjuntos *s* em *NewSubsets*2, *x* não pertence a *s*. Para ilustrar este passo, suponha que são considerados apenas subconjuntos com duas soluções. Neste caso, *NewSubsets*1 = $\{(y^1, x), (y^2, x), (y^3, x)\}$ e *NewSubsets*2 = $\{(y^1, y^2), (y^1, y^3), (y^2, y^3)\}$. Um subconjunto *s* não pode ser admitido em *NewSubsets*2 se já foi usado numa combinação na iteração corrente. Por exemplo, se (y^1, y^2) já tivesse sido combinado nesta mesma iteração, então este subconjunto não deveria ser incluído em *NewSubsets*2. Os elementos de *NewSubsets* são ordenados de forma a permitir que os subconjuntos de soluções sejam considerados para combinação em ordem crescente de cardinalidade. Por exemplo, o método de combinação é aplicado primeiro nos subconjuntos que têm duas soluções, depois nos subconjuntos formados por três soluções, e assim sucessivamente. Para subconjuntos de soluções com a mesma cardinalidade, os elementos de *NewSubsets*1 estão disponíveis para combinação antes dos

elementos de *NewSubsets*2, e dentre os elementos de *NewSubsets*1 e *NewSubsets*2, os elementos são ordenados em ordem lexicográfica. Por exemplo, a ordem para considerar os elementos de *NewSubsets* para uma combinação seria *NewSubsets* = $\{(y^1,x), (y^2,x), (y^3,x), (y^1, y^2), (y^1, y^3), (y^2, y^3)\}$. Se uma nova solução tentativa gerada no Passo 3.3 não é adicionada a *RefSet,* o Passo 3.8 elimina o subconjunto *s* de *NewSubsets.* O procedimento pára quando *NewSubsets* está vazio, ou se o número máximo de soluções avaliadas for atingido. Um exemplo completo de uma iteração do *scatter search* é apresentado no final deste capítulo.

Passo 3. Combinação com atualização dinâmica

Passo 3.1. Geração de subconjuntos. Gere *NewSubsets.* Faça *NewSolutions* = FALSE. *NewSubsets*1 =∅.

enquanto (*NewSubsets* ≠ ∅)) e (número de soluções avaliadas < *MaxSol*) **faça {**

Passo 3.2. Selecione o próximo subconjunto *s* em *NewSubsets*

Passo 3.3. Aplique o método de combinação em *s* para obter uma nova solução tentativa *x*. *x*←Heurística_de_Melhoria (*x*).

se atualização por qualidade e diversidade **então {**

se *x* tem qualidade melhor do um elemento *RefSet*1 **então{**

Passo 3.4. Inclua *x* em *RefSet*1, e elimine o pior elemento, em termos de valor de função objetivo, de *RefSet*1. Faça *NewSolutions* = TRUE.

} caso contrário se *x* aumentar a distância mínima dos elementos em *RefSet*2 **{**

Passo 3.5. Inclua *x* em *RefSet*2, elimine o elemento com menor valor de distância mínima. Faça *NewSolutions* = TRUE.

}

} caso contrário se atualização por qualidade somente, **então {**

se *x* tem uma qualidade melhor do que o pior elemento de *RefSet* **então {**

Passo 3.6 Inclua *x* em *RefSet*, elimine o pior elemento de *RefSet* em termos de função objetivo. Faça *NewSolutions* = TRUE.

}

}

se (*NewSolutions* = TRUE) **então {**

Passo 3.7. Construa *NewSubsets*1 que consiste de subconjuntos de soluções geradas pelo método de geração de subconjuntos, no qual, para todos os subconjuntos *s* em *NewSubsets*1, *x* é um elemento de *s*. Construa *NewSubsets*2 que consiste de subconjuntos de soluções geradas pelo método de geração de subconjuntos, no qual, para todos os subconjuntos *s* em *NewSubsets*2*, x* não pertence a *s*, e *s* ainda não foi utilizado numa combinação. Faça *NewSolutions* = FALSE e *NewSubsets* = *NewSubsets*1 ∪ *NewSubsets*2. Os elementos de *NewSubsets* são ordenados obedecendo as seguintes regras: 1) os subconjuntos de soluções são colocados em ordem crescente de cardinalidade, 2) entre subconjuntos de mesma cardinalidade, os elementos pertencentes a *NewSubsets*1 são combinados primeiro, em ordem lexicográfica, e os elementos pertencentes a *NewSubsets*2 são posicionados em *NewSubsets* após os elementos de *NewSubsets*1, também em ordem lexicográfica.

} caso contrário {

Passo 3.8. Elimine *s* de *NewSubsets.*

}

}

Figura 1.8 Combinação com atualização dinâmica

1.3.3 Método de geração de subconjuntos

O método de geração de subconjuntos proposto no modelo de Glover (1998) é utilizado aqui. Quatro tipos de subconjuntos podem ser gerados:

1) *Subconjunto Tipo* 1: Todos os subconjuntos de 2-soluções.
2) *Subconjunto Tipo* 2: Subconjuntos de 3-soluções derivados dos subconjuntos de 2-soluções, obtidos ao incluir a melhor solução (medida pelo valor de função objetivo) no subconjunto de 2-soluções, se esta solução não estiver presente neste subconjunto.
3) *Subconjunto Tipo* 3: Subconjuntos de 4-soluções, obtidos ao incluir em cada subconjunto de 3-soluções, a melhor solução que não esteja neste subconjunto.
4) *Subconjunto Tipo* 4: Os subconjuntos que consistem das *i* melhores soluções para $i = 5$ até *b*.

1.3.4 Método de combinação de soluções

Seja $s = (a^1,...,a^r)$ um subconjunto com *r* soluções, pertencente a *NewSubsets*. O objetivo deste método é construir uma nova solução a partir da combinação das *r* soluções de *s*. Para isso, diversos métodos de combinação são propostos abaixo.

Método de Combinação A: Neste método, uma combinação de soluções é dada por,

$$a'_k = \left\lfloor \frac{\sum_{j=1,...,r} v_j \cdot a_k^j}{\sum_{j=1,...,r} v_j} \right\rfloor \qquad k = 1,...,m$$

$$v_j = \frac{1}{VO(j)}$$

tal que $VO(j)$ é o valor da função objetivo da solução j e $\lfloor x \rfloor$ denota o maior inteiro menor ou igual a *x*. O objetivo deste tipo de combinação é dar um peso maior para as soluções que apresentam um melhor valor de função objetivo.

Método de Combinação B: Similar à Combinação A, mas os pesos v_j são aleatórios.

Método de Combinação C: Combinação que faz uso da matriz de freqüência *M*, definida no método de geração de soluções diversas. A idéia é favorecer valores de recursos que são menos freqüentes durante a busca, e desta forma, permitir um grau de diversificação. Para cada solução *j* = 1,...,*r* pertencente a *s*, e para cada tipo de recurso *k* = 1,..., *m*, $v_{jk} = M[k][i]$, tal que *i* é o subintervalo tal que $z_k^i \le a_k^j < z_k^{i+1}$, então a combinação é dada por,

$$a_k^{'} = \left\lfloor \frac{\sum_{j=1,...,r} (1/v_{jk}) \cdot a_k^j}{\sum_{j=1,...,r} (1/v_{jk})} \right\rfloor \qquad k = 1,...,m$$

Método de Combinação D: Calcula o valor máximo

$$a_k^{'} = \max_{j=1,...,r} \{a_k^j\} \qquad k = 1,...,m$$

Método de Combinação E: Calcula o valor mínimo

$$a_k^{'} = \min_{j=1,...,r} \{a_k^j\} \qquad k = 1,...,m$$

Método de Combinação F: O objetivo desta combinação é selecionar a melhor solução *t*, dentre as *r* a serem combinadas, e fazer uma pequena perturbação em *t*. Para um dado recurso *k*, sejam $N_1 = \{j \in s \mid a_k^j > a_k^t\}$, $N_2 = \{j \in s \mid a_k^j < a_k^t\}$, e $N_3 = \{j \in s \mid a_k^j = a_k^t\}$, os conjuntos de soluções com valores de disponibilidade de recursos maior, menor ou igual a a_k^t, respectivamente.

- Se $Card(N_1) > \max(Card(N_2), Card(N_3))$, $a_k^{'} = a_k^t + 1$.
- Se $Card(N_2) > \max(Card(N_1), Card(N_3))$, $a_k^{'} = a_k^t - 1$
- Se nenhuma das condições acima for verdade, então: $a_k^{'} = a_k^t$

O seguinte exemplo ilustra o método de combinação F. Considere um projeto com 4 tipos de recursos e um subconjunto *s* com três soluções, *s* = {(1, 2, 4, 5), (3, 2, 10, 4), (1, 2, 3, 6)}, composto pelas soluções 1, 2 e 3, nesta ordem. Suponha que *VO*(1) = 10, *VO*(2) = 5, *VO*(3) = 5.

As soluções 2 e 3 possuem o menor valor de função objetivo. Suponha que a solução a^2 = (3, 2, 10, 4), $VO(2) = 5$, seja escolhida como melhor solução. Para $k = 1$, $a_1^1 = 1$ e $a_1^3 = 1$ são menores do que $a_1^2 = 3$, portanto $a_1^{'} = 3\text{-}1 = 2$. Para $k = 2$, as soluções 1 e 3 têm a mesma disponibilidade de recursos que a solução 2, portanto, $a_2^{'} = 2$. Para $k = 3$, a_3^1 e a_3^3 são menores do que $a_3^2 = 10$, portanto, $a_3^{'} = 10 - 1 = 9$. Para $k = 4$, a_4^1 e a_4^3 são maiores do que $a_4^2 = 4$, portanto, $a_4^{'} = 4 + 1 = 5$.

Método de Combinação G: Combinação aleatória uniforme, a combinação das r soluções é dada por,

$$a_k^{'} = a_k^j \qquad k = 1,...,m$$

no qual j é obtido, para cada valor de k, por uma distribuição uniforme discreta U[1, r].

Método de Combinação H: Este método de combinação seleciona probabilisticamente, uma das combinações (A-G) descritas acima, de acordo com seu desempenho ao longo da busca. Soluções de *RefSet*1 são ordenadas em ordem crescente de valor da função objetivo, sendo que a melhor solução ocupa a primeira posição em *RefSet*1. Quando a solução obtida pelo método de combinação p qualifica-se para ocupar a j-ésima posição em *RefSet*1, o valor b_1-j+1 é adicionado a *score(p)*. Portanto, um método de combinação que resulta em soluções de alta qualidade apresenta pontuação mais alta, e tem maior probabilidade de ser selecionado. No início da busca, a seleção do método de combinação é realizada aleatoriamente. A pontuação será utilizada somente após *InitIter* combinações terem sido executadas.

1.4. Heurística de múltiplos inícios (*multi-start*)

Nesta seção, duas versões da heurística de múltiplos inícios são propostas. O objetivo, ao implementar estas heurísticas, é avaliar o ganho de qualidade promovido pelos procedimentos *scatter search* com respeito a heurísticas de implementação mais simples ou que não utilizam estratégias tão avançadas como o *scatter search*.

Os principais passos da heurística de múltiplos inícios são descritos abaixo.

Passo 1. Gerar os valores de recursos a_k, $k = 1,..., m$.
Passo 2. Aplicar a heurística de melhoria.
Passo 3. Repetir passos 1 e 2 até que o número de soluções avaliadas atinja *MaxSol*.

Duas versões da heurística múltiplos inícios são desenvolvidas, heurística de múltiplos inícios aleatória (MSA) e heurística de múltiplos inícios baseada em freqüência (MSF). A diferença entre MSA e MSF está no Passo 1. Em MSA, os valores a_k, $k = 1,..., m$, são gerados por uma distribuição uniforme discreta $U[l_k , u_k]$. Em MSF, estes valores são dados pelo método de geração de soluções diversas (Figura 1.9), que utiliza a história da busca para gerar soluções diversas.

procedimento MSF ($\lambda, g, MaxSol$ **)**

Passo 0.

Calcule z_k^g : gere uma solução *a* através de uma distribuição uniforme U(l_k , u_k).

H←Heurística_de_Melhoria (*a*), e obtenha uma solução $H = (h_1,...,h_m)$.

Se solução *H* for factível, então

{

Faça $z_k^g = \lambda \cdot h_k$, k = 1, ..., m. Inicialize $M[k][i] = 1$, k = 1,..., m, i = 0,.., g-1. Vá para o **Passo 1**.

}caso contrário, repita o **Passo 0**.

Passo 1. Para cada recurso k = 1,..., m, gere valores para a_k : primeiro, selecione um subintervalo com probabilidade inversamente proporcional a $M[k][i]$, e então, gere aleatoriamente um valor para a_k , dentro do subintervalo selecionado.

Passo 2. Aplique *a*←Heurística_de_Melhoria (*a*).

Passo 2.1. Atualize a matriz de freqüência *M*.

Passo 3. Repita os **Passos 1** e **2**, até que o número de soluções avaliadas atinja *MaxSol*

Figura 1.9 Procedimento MSF

1.5 Exemplo do método *scatter search*

Nesta seção, um exemplo é apresentado para ilustrar uma iteração do *scatter search*. O exemplo consiste de um projeto com 30 atividades, 4 tipos de recursos e data de entrega $D = 53$. As relações de precedência entre as atividades são descritas na Tabela 1.3. O valor da função objetivo é calculado por: $VO = a_1c_1 + a_2c_2 + a_3c_3 + a_4c_4$, e a Tabela 1.4 indica o valor do custo relativo à disponibilidade de um determinado recurso. As Tabelas 1.5 e 1.6 descrevem a quantidade de recursos que uma atividade precisa para ser processada e a sua duração, respectivamente. Neste exemplo, os parâmetros utilizados para o *scatter search* são $PSize = 10$, $b = 5$, $b_1 = 3$, $b_2 = 2$, $g = 10$ e $\lambda = 1{,}9$. A atualização do conjunto de referência é realizada de forma dinâmica e por qualidade. Os 4 tipos de subconjuntos de soluções descritos na Seção 1.3.3 são utilizados, e as combinações são realizadas com o método de combinação G.

Tabela 1.3 Relações de precedência entre as atividades

Atividade	Sucessores		Atividade	Sucessores			Atividade	Sucessores		
1			**12**	5	10		**23**	7	14	18
2	1		**13**	10			**24**	9	21	22
3	1		**14**	2			**25**	8	19	21
4	1		**15**	7	13	14	**26**	18	22	24
5	2		**16**	7	9	14	**27**	6	11	23
6	4	5	**17**	9	11	15	**28**	23	24	25
7	5		**18**	12	13		**29**	17	21	27
8	2	3	**19**	6	12	15	**30**	16	26	28
9	8		**20**	11	13		**31**	3	18	19
10	4		**21**	12	15	20	**32**	29	30	31
11	4		**22**	3	19	20				

Tabela 1.4 Custo dos recursos

c_1	c_2	c_3	c_4
1	9	4	9

Tabela 1.5 Recursos utilizados pelas atividades

Atividade	r_1	r_2	r_3	r_4	Atividade	r_1	r_2	r_3	r_4	Atividade	r_1	r_2	r_3	r_4
1	0	0	0	0	**11**	8	5	3	3	**22**	4	5	10	6
2	3	9	1	9	**12**	1	6	8	2	**23**	8	5	5	7
3	8	8	3	10	**13**	1	9	10	7	**24**	4	4	4	2
4	3	4	3	1	**14**	5	8	2	5	**25**	4	10	7	8
5	1	7	6	4	**15**	3	10	4	9	**26**	4	10	3	1
6	1	2	5	4	**16**	8	9	10	7	**27**	10	10	7	1
7	7	5	7	10	**17**	9	8	8	5	**28**	7	3	6	6
8	6	3	1	3	**18**	8	2	6	8	**29**	5	1	10	7
9	1	8	7	6	**19**	8	1	7	9	**30**	3	2	4	5
10	2	9	3	4	**20**	6	8	5	4	**31**	7	3	2	8
					21	5	3	4	10	**32**	0	0	0	0

Tabela 1.6 Duração das atividades

Atividade	Duração	Atividade	Duração	Atividade	Duração	Atividade	Duração	Atividade	Duração
1	0	**8**	7	**15**	3	**21**	4	**27**	8
2	2	**9**	2	**16**	5	**22**	5	**28**	7
3	4	**10**	7	**17**	6	**23**	5	**29**	9
4	4	**11**	2	**18**	1	**24**	8	**30**	6
5	1	**12**	3	**19**	9	**25**	4	**31**	2
6	10	**13**	4	**20**	7	**26**	3	**32**	0
7	10	**14**	3						

A Tabela 1.7 lista as soluções que compõem o conjunto *P*, gerado no Passo 1 do *scatter search* (Figura 1.5), e o valor da função objetivo correpondente a cada solução. O próximo passo do *scatter search* é construir o conjunto de referência, e para isso, as $b_1 = 3$ soluções de *P* com menor valor de função objetivo são selecionadas: 5, 4 e 8. Estas são as soluções de alta qualidade. Para selecionar as $b_2 = 2$ soluções diversas de *P*, é necessário calcular o valor das distâncias das soluções que já estão em *RefSet*, em relação às demais soluções que encontram-se em *P*. Estas distâncias são descritas pela Tabela 1.8.

Tabela 1.7 População inicial

Solução	a_1	a_2	a_3	a_4	*VO*
1	20	24	30	23	563
2	22	24	23	25	555
3	21	24	30	24	573
4	22	24	23	24	546
5	23	24	22	23	534
6	22	24	25	24	554
7	22	27	22	23	560
8	21	24	26	23	548
9	20	25	30	23	572
10	21	24	30	23	564

Tabela 1.8 Distâncias entre as soluções

Soluções	**6**	**2**	**7**	**1**	**10**	**9**	**3**
5	5	4	4	11	10	12	11
4	2	1	5	10	9	11	8
8	3	6	8	5	4	6	5
Menor distância	2	1	4	5	4	6	5

Tabela 1.9 Distâncias entre as soluções

Soluções	**6**	**2**	**7**	**1**	**10**	**3**
5	5	4	4	11	10	11
4	2	1	5	10	9	8
8	3	6	8	5	4	5
9	9	12	12	1	2	3
Menor distância	2	1	4	1	2	3

As distâncias descritas na Tabela 1.8 são calculadas pela fórmula de distância definida na Seção 1.3.2. Por exemplo, a distância entre as soluções 5 e 6 é dada por:

$$d(5, 6) = |23\text{-}22| + |24\text{-}24| + |22\text{-}25| + |23\text{-}24| = 5$$

A última linha da Tabela 1.8 mostra a distância mínima de uma dada solução no conjunto *P* (listada na primeira linha da tabela) em relação às soluções de *RefSet* (primeira coluna da tabela). Note que a solução 9 apresenta a maior distância mínima, portanto, esta solução é inserida no

conjunto de referência, e as distâncias são recalculadas. A Tabela 1.9 exibe os novos valores de distância, após a solução 9 ter sido incluída em *RefSet*. É possível observar que a solução 7 maximiza a menor distância, e portanto, a solução 7 é incluída em *RefSet*. Desta forma, *RefSet*1 = {5, 4, 8} e *RefSet*2 = {9, 7}. A Tabela 1.10 ilustra as soluções que compõem *RefSet* e os respectivos valores de função objetivo (*VO*).

Tabela 1.10 Conjunto de referência inicial

RefSet	a_1	a_2	a_3	a_4	***VO***
1	23	24	22	23	534
2	22	24	23	24	546
3	21	24	26	23	548
4	22	27	22	23	560
5	20	25	30	23	572

Após construir o conjunto de referência, os subconjuntos de soluções são gerados e adicionados a *NewSubsets* em ordem lexicográfica (Passo 3.1 do *scatter search* – Figura 1.8). Desta forma, *NewSubsets* contém os seguintes subconjuntos de soluções: *NewSubsets* = {(1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5), (2,3,1), (2,4,1), (2,5,1), (3,4,1),(3,5,1), (4,5,1), (3,4,1,2), (3,5,1,2), (4,5,1,2), (1,2,3, *best*4, *best*5)}, no qual *best*4 e *best*5 são a quarta e quinta melhores soluções encontradas até o momento, respectivamente. Note que as duas melhores soluções encontradas até o momento são as soluções 1 e 2 de *RefSet.* A Tabela 1.11 mostra a solução resultante (x) da combinação de (1,2), assim como a solução obtida após aplicar a heurística de melhoria em x, e o valor da função objetivo da solução melhorada (Passo 3.3 – Figura 1.8). O valor da função objetivo (*VO* = 587) é maior do que o pior elemento de *RefSet*, então esta solução é descartada, e prossegue-se para a combinação do próximo subconjunto. A combinação da solução 1 com a solução 3, e subseqüente aplicação da heurística de melhoria, gera uma solução cujo valor da função objetivo é menor do que a pior solução de *RefSet*, como mostrado na última linha da Tabela 1.11. Como a atualização é dinâmica, a solução combinada é inserida no conjunto de referência imediatamente, e deve ser combinada o mais rápido possível. A solução de pior valor de função objetivo é eliminada de *RefSet*, e a nova solução combinada é inserida em *RefSet* (Passo 3.6 – Figura 1.8). O conjunto de referência atualizado pode ser visto na

Tabela 1.12. Note que, se o tipo de atualização fosse estático, todas as combinações seriam avaliadas antes do conjunto de referência ser atualizado.

Tabela 1.11 Resultado das combinações de soluções

s	**Solução combinada**				**Solução combinada após melhoria**				***VO***
	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	
(1,2)	23	24	22	23	22	27	22	26	587
(1,3)	23	24	26	23	23	24	26	23	541

Tabela 1.12 Reconstrução do conjunto de referência.

RefSet	a_1	a_2	a_3	a_4	***VO***	
1	23	24	22	23	534	
2	23	24	26	23	541	Novo
3	22	24	23	24	546	
4	21	24	26	23	548	
5	22	27	22	23	560	

O próximo passo (Passo 3.7 – Figura 1.8) constrói os conjuntos *NewSubsets*1 e *NewSubsets*2. *NewSubsets*1 contém todos os subconjuntos de soluções, nos quais uma das soluções é a solução que acabou de entrar em *RefSet*, isto é, *NewSubsets*1= {(1,2), (2,3), (2,4), (2,5), (2,3,1), (2,4,1), (2,5,1), (3,4,1,2), (3,5,1,2), (4,5,1,2), (1,2,3, *best*4, *best*5)}. *NewSubsets*2 contém o restante dos subconjuntos que não pertencem a *NewSubsets*1 e que não foram combinadas nesta iteração ainda, isto é, *NewSubsets*2 = {(1,5), (3,4), (3,5), (4,5),(3,4,1),(3,5,1), (4,5,1)}. Finalmente, *NewSubsets* = {(1,2), (2,3), (2,4), (2,5), (1,5), (3,4), (3,5), (4,5), (2,3,1), (2,4,1), (2,5,1), (3,4,1), (3,5,1), (4,5,1), (3,4,1,2), (3,5,1,2), (4,5,1,2), (1,2,3, *best*4, *best*5)} As soluções 1 e 2, da Tabela 1.12, são combinadas e dão origem a uma nova solução combinada com *VO* = 530 (Tabela 1.13). Este valor é menor do que a pior solução de *RefSet*, portanto, *RefSet* é atualizado, como mostrado na Tabela 1.14.

Tabela 1.13 Resultado da combinação de soluções

s	**Solução combinada**				**Solução combinada após melhoria**				***VO***
	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	
(1,2)	23	24	22	23	21	24	26	21	530

Tabela 1.14 Reconstrução do conjunto de referência.

RefSet	a_1	a_2	a_3	a_4	***VO***	
1	21	24	26	21	530	novo
2	23	24	22	23	534	
3	23	24	26	23	541	
4	22	24	23	24	546	
5	21	24	26	23	548	

As soluções 1 e 2, da Tabela 1.14, são combinadas e dão origem a uma nova solução combinada com *VO* = 543 (Tabela 1.15). Este valor é menor do que a pior solução de *RefSet*, portanto, *RefSet* é atualizado, como mostrado na Tabela 1.16.

Tabela 1.15 Resultado da combinação de soluções

s	**Solução combinada**				**Solução combinada após melhoria**				***VO***
	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	
(1,2)	21	24	22	21	23	27	22	21	543

Tabela 1.16 Reconstrução do conjunto de referência

RefSet	a_1	a_2	a_3	a_4	***VO***	
1	21	24	26	21	530	
2	23	24	22	23	534	
3	23	24	26	23	541	
4	23	27	22	21	543	novo
5	22	24	23	24	546	

As soluções do conjunto de referência da Tabela 1.16 são combinadas e geram as soluções exibidas na Tabela 1.17. A combinação das soluções 4 e 5 gera uma solução de menor valor de função objetivo do que *RefSet*[5], logo, *RefSet* é reconstruído, como mostrado na Tabela 1.18.

Tabela 1.17 Resultado da combinação de soluções

s	Solução combinada				Solução combinada após melhoria				*VO*
	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4	
(1,4)	21	24	22	21	21	24	24	26	567
(2,4)	23	27	22	23	22	24	25	27	554
(3,4)	23	24	26	22	22	26	24	23	559
(4,5)	23	24	22	21	22	24	22	23	533

Tabela 1.18 Reconstrução do conjunto de referência

RefSet	a_1	a_2	a_3	a_4	*VO*	
1	21	24	26	21	530	
2	22	24	22	23	533	novo
3	23	24	22	23	534	
4	23	24	26	23	541	
5	23	27	22	21	543	

As soluções do conjunto de referência da Tabela 1.18 são combinadas e geram as soluções exibidas na Tabela 1.19. A Tabela 1.19 mostra os subconjuntos de soluções, na ordem em que foram combinados, e o valor da função objetivo após a combinação e aplicação da heurística de melhoria. A combinação das soluções 1 e 4 geram uma solução de menor valor de função objetivo do que *RefSet*[5], logo, *RefSet* é reconstruído, como mostrado na Tabela 1.20.

Tabela 1.19 Resultado da combinação de soluções

Ordem de comb.	*s*	*VO*	Ordem de comb.	*s*	*VO*
1	(1,2)	543	4	(2,5)	543
2	(2,3)	534	5	(1,4)	537
3	(2,4)	587			

Tabela 1.20 Reconstrução do conjunto de referência

RefSet	a_1	a_2	a_3	a_4	*VO*	
1	21	24	26	21	530	
2	22	24	22	23	533	
3	23	24	22	23	534	
4	22	24	23	23	537	novo
5	23	24	26	23	541	

As soluções do conjunto de referência da Tabela 1.20 são combinadas e geram as soluções exibidas na Tabela 1.21. A Tabela 1.21 mostra os subconjuntos de soluções, na ordem em que foram combinados, e o valor da função objetivo após a combinação e aplicação da heurística de melhoria. Nenhuma dessas combinações gerou uma solução melhor do que *RefSet*[5], e neste ponto, *NewSubsets* está vazio, portanto, uma iteração foi completada. O próximo passo seria reconstruir o conjunto de referência (Passo 4 – Figura 1.5), se o critério de parada ainda não tiver sido cumprido.

Tabela 1.21 Resultado da combinação de soluções

Ordem de comb.	*s*	*VO*	**Ordem de comb.**	*s*	*VO*	**Ordem de comb.**	*s*	*VO*
1	(1,4)	559	5	(2,4,1)	550	11	(3,4,1,2)	573
2	(2,4)	537	6	(3,4,1)	546	12	(4,5,1,2)	537
3	(3,4)	573	7	(4,5,1)	573	13	(3,5,1,2)	537
4	(4,5)	537	8	(2,3,1)	573	14	(1,2,3, *best*4,*best*5)	555
			9	(2,5,1)	542			
			10	(3,5,1)				

CAPÍTULO 2

EXPERIMENTOS COMPUTACIONAIS PARA O PCDR

Este capítulo apresenta os experimentos computacionais para o problema de custo de disponibilidade de recursos. Para problemas testes de pequeno porte, os procedimentos propostos são comparados com soluções ótimas obtidas pelo método de Demeulemeester (1995), e limitantes inferiores e superiores de (Drexl e Kimms, 2001). Para problemas testes de médio porte, foram desenvolvidas heurísticas simples, de múltiplos inícios, com o intuito de mostrar o ganho de qualidade gerado pelo *scatter search*.

2.1 Experimentos computacionais

Nesta seção, o desempenho das versões do método *scatter search* proposto é avaliado em três conjuntos de problemas testes para o PCDR. O Conjunto 1 consiste de problemas de médio porte, que foram utilizados para analisar o desempenho das versões do *scatter search* e a influência de parâmetros. O Conjunto 2 consiste de problemas testes maiores, que foram utilizados para comparar o desempenho das melhores versões dinâmica e estática do *scatter search*. O Conjunto 3 consiste de problemas testes menores gerados por Drexl e Kimms (2001). Os autores também fornecem limitantes superiores e inferiores para o PCDR. Além destes limitantes fornecidos por Drexl e Kimms (2001), o algoritmo de plano de corte de Demeulemeester (1995) foi implementado, e portanto, soluções exatas e limitantes inferiores

gerados por este método foram adicionados ao Conjunto 3. Experimentos computacionais foram realizados num PC Pentium III, 667MHz, 256 Mbyte RAM. Todos os procedimentos foram escritos na linguagem de programação C++.

2.2 Geração de problemas testes

O PCDR e o PPPRL possuem alguns parâmetros em comum, isto torna possível adaptar, de forma relativamente simples, problemas testes do PPPRL já existentes, para o PCDR. O programa Progen (Kolisch *et al.*, 1995) é um gerador de problemas testes para o PPPRL, que foi utilizado para gerar os problemas testes da biblioteca PSPLIB (http://www.bwl.uni-kiel.de/Prod/psplib/), que envolvem 4 tipos de recursos e 30, 60, 90 e 120 atividades. Neste trabalho, testes computacionais são realizados com problemas testes gerados pelo Progen e adaptados para o PCDR seguindo a metodologia de Drexl e Kimms (2001). Problemas testes com 4 recursos e 30, 60, 90 e 120 atividades são diretamente obtidos da biblioteca PSPLIB. O restante dos problemas testes foram gerados pelo programa Progen. Dois importantes parâmetros do Progen são a complexidade de rede (*network complexity – NC*) e o fator de recursos (*resource factor – RF*). *NC* reflete o número médio de sucessores imediatos de uma atividade. O fator de recurso varia entre [0,1] e reflete a densidade dos diferentes tipos de recursos que uma atividade requer para ser processada. Por exemplo, se $RF = 1$, 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 e Kimms (2001) calculam a data de entrega para o projeto como uma função do caminho crítico do projeto,

$$D = DF \cdot \max EF_i$$

no qual DF é o fator de data de entrega e EF_i é o instante de início mais cedo da atividade i. Para cada problema teste, os custos C_k são gerados por uma distribuição uniforme U[1,10].

2.3 Resultados computacionais para o Conjunto 1

O Conjunto 1 envolve problemas com $n = 30, 45, 60$ e 75 atividades e 4 tipos de recursos, com os seguintes valores de parâmetros:

- Fator de recurso (*RF*): 0,25 e 0,75.
- Complexidade de rede (*NC*): 1,5
- Fator de data de entrega (*DF*): 1,2.

Para cada combinação de *RF, DF, NC, n* e *m*, 5 problemas testes foram gerados, resultando num total de 40 problemas testes. O objetivo dos testes realizados com o Conjunto 1 é analisar o desempenho relativo das versões *scatter search* (SS) e heurísticas de múltiplos inícios (MSA e MSF). Para todas as versões do *scatter search* utilizaram-se os seguintes parâmetros: método de combinação A, *Psize* = 30, g = 10 e λ = 1,9. 12 versões do *scatter search* foram geradas combinando os seguintes fatores:

- Três valores para (b_1, b_2): (5,5), (7,3) e (3,7)
- Critérios de atualização do *RefSet*: *QD* ou *Q*.
- Freqüência de atualização do *RefSet*: *Estática* ou *Dinâmica*

Cada versão é denotada por três campos, F_1/ F_2 / F_3, nos quais F_1 = (b_1, b_2), F_2 = *QD* ou *Q*, F_3 = *Estático* ou *Dinâmico*. O critério de parada para os métodos de múltiplos inícios e *scatter search* foi definido da seguinte forma. Seja SS0 a versão do *scatter search* com (5, 5)/ *QD*/ *Estático*. O critério de parada para SS0 foi 5 iterações, isto é, *MaxIter* = 5, com um limite de tempo de uma hora. O número de soluções avaliadas para esta versão do *scatter search* foi adotado como critério de parada para as outras versões dos procedimentos de múltiplos inícios e *scatter search*. Este número, mostrado na Tabela 2.1, é obtido como um número médio de soluções avaliadas em 10 problemas testes.

Tabela 2.1 Número médio de soluções avaliadas.

	n			
	30	**45**	**60**	**75**
Média	973097	1249603	1347018	1605204

Seja distância = 100(*H-Best*)/(*Best*), no qual *H* é o valor da função objetivo da solução encontrada pela heurística e *Best* é o melhor valor de solução conhecido, para um dado problema teste. O desempenho das heurísticas propostas é avaliado pelas seguintes medidas:

- Distância média
- Distância máxima
- Número de vitórias: número de problemas testes nos quais *H* = *Best*

Tabela 2.2 Distância média, distância máxima e número de vitórias para SS, MSA e MSF

Parâmetros			***n***				**Distância média (%)**	**Distância máx. (%)**	**Núm. de vitórias**
			30	**45**	**60**	**75**			
$b_1 = 5$, $b_2 = 5$	**QD**	***Estático***	1,29	0,73	1,36	1,07	1,11	4,02	6
		Dinâmico	2,28	0,89	1,50	1,00	1,42	9,63	6
	Q	***Estático***	1,72	0,89	1,16	1,00	1,19	6,16	6
		Dinâmico	1,29	0,77	1,28	0,91	1,06	4,02	9
$b_1 = 3$, $b_2 = 7$	**QD**	***Estático***	1,13	0,83	1,27	1,21	1,11	5,82	8
		Dinâmico	1,59	1,33	1,78	0,89	1,40	6,27	7
	Q	***Estático***	1,33	1,00	1,52	1,26	1,28	5,20	7
		Dinâmico	1,29	1,01	1,19	1,08	1,14	4,02	5
$b_1 = 7$, $b_2 = 3$	**QD**	***Estático***	1,87	0,75	1,50	1,19	1,33	9,63	6
		Dinâmico	1,39	0,48	1,36	0,86	1,02	5,04	11
	Q	***Estático***	2,19	0,80	1,44	0,96	1,35	9,63	7
		Dinâmico	1,44	0,95	1,19	1,13	1,18	4,02	7
MSA			4,12	6,18	7,49	7,50	6,32	15,10	1
MSF			2,83	2,46	3,30	3,16	2,94	9,08	1

Para cada valor de *n*, a Tabela 2.2 mostra a distância média para 10 problemas testes. Os números das últimas três colunas foram calculados com respeito a 40 problemas testes. Todas as versões do *scatter search* tiveram um desempenho superior às heurísticas de múltiplos inícios,

em termos de distância média e número de vitórias. Como poderia ser esperado, a heurística MSF foi superior à heurística MSA, graças à memória baseada em freqüência que foi utilizada em MSF para gerar as soluções. Dentre as versões do *scatter search* testadas, não existe uma versão dominante, em termos de distância média, para todos os valores de n. Em média, entretanto, a versão do *scatter search* com parâmetros (7, 3)/ *QD*/ *Dinâmico* apresentou a menor distância média e o maior número de vitórias, seguido pela versão (5, 5)/ *Q*/ *Dinâmico*. Dentre os valores de b_1 e b_2 testados, nenhum par de valores foi consistentemente superior aos demais. É possível notar, na tabela, que a atualização dinâmica sempre tem um desempenho médio melhor do que a estática, quando a atualização é do tipo *Q*.

Tabela 2.3 Número médio de inícios

(b_1,b_2)	***Estático***		***Dinâmico***	
	QD	***Q***	***QD***	***Q***
(5,5)	5,00	5,23	7,88	7,88
(7,3)	5,58	5,35	7,70	8,20
(3,7)	5,78	5,03	7,70	7,73

A Tabela 2.3 mostra o número médio de vezes em que o método de geração de soluções diversas foi utilizado para reconstruir o conjunto de referência, isto é, o número médio de inícios. Note que a atualização estática apresentou um número de inícios menor do que a atualização dinâmica, confirmando portanto, que a estratégia dinâmica converge mais rapidamente do que a estática. A escolha da estratégia dinâmica ou estática depende, muito provavelmente, do problema abordado. No problema estudado neste trabalho, a atualização dinâmica, que introduz um número maior de soluções diversas ao longo da busca, parece ser uma estratégia mais eficiente do que a estratégia estática, que gasta mais tempo em regiões induzidas por menos reconstruções do conjunto de referência.

Tabela 2.4 Porcentagem da distribuição das melhores soluções encontradas pela combinação de soluções de *RefSet* e encontradas em *P*

Método de Comb.	$b_1 = 5, b_2 = 5$				$b_1 = 3, b_2 = 7$				$b_1 = 7, b_2 = 3$				**Média (%)**
	QD		**Q**		***QD***		***Q***		***QD***		***Q***		
	Estát.	***Din.***	***Estát.***	***Din.***	***Estát.***	***Din.***	***Estát.***	***Din.***	***Estát.***	***Din.***	***Estát.***	***Din.***	
P	2,5	10,0	7,5	7,5	5,0	7,5	7,5	7,5	5,0	15,0	20,0	12,5	8,96
***RefSet*1**	32,5	40,0	25,0	25,0	25,0	25,0	10,0	7,5	47,5	32,5	50,0	55,0	31,25
***RefSet*1& *RefSet*2**	60,0	32,5	50,0	50,0	40,0	40,0	37,5	55,0	45,0	47,5	27,5	27,5	42,71
***RefSet*2**	5,0	17,5	17,5	17,5	30,0	27,5	45,0	30,0	2,5	5,0	2,5	5,0	17,08

Desconsiderando a heurística de melhoria, o *scatter search* gera soluções que podem ser provenientes de uma combinação de soluções em *RefSet*1, uma combinação de soluções em *RefSet*2, uma combinação envolvendo soluções tanto de *RefSet*1 como *RefSet*2, ou ainda proveniente do conjunto *P* (gerado pelo método de geração de soluções diversas). A Tabela 2.4 mostra que a maior parte das melhores soluções originam-se da combinação de soluções de alta qualidade somente, ou de combinações envolvendo soluções de alta qualidade e soluções diversas. A contribuição do conjunto de soluções diversas é mais significativa para versões com $b_1 = 3$ e $b_2 = 7$.

A Tabela 2.5 mostra o tempo médio gasto pelos métodos *scatter search* e de múltiplos inícios, assim como o tempo médio gasto para encontrar a melhor solução. Note que a heurística MSA é incapaz de melhorar soluções encontradas no início da busca, ao contrário do que ocorre como as heurísticas *scatter search* e MSF.

Tabela 2.5 Tempo computacional médio (em segundos)

	n			
	30	**45**	**60**	**75**
SS	448,69	1025,94	1680,87	2782,20
SS melhor sol.	139,23	382,56	823,45	1212,36
MSF	429,59	985,01	1650,78	2797,10
MSF melhor sol.	60,63	331,88	344,30	1016,65
MSA melhor sol.	9,31	12,03	19,33	52,18

No próximo experimento, foi testada a influência dos métodos de combinação propostos na Seção 1.3.4 do Capítulo 1. As versões do *scatter search* (5, 5)/ *QD/ Estático,* (7, 3)/ *QD*/ *Dinâmico* e (5, 5)/ *Q/ Dinâmico* foram utilizadas para este teste. As Tabelas 2.6 e 2.8 mostram que a combinação aleatória do tipo G apresentou o melhor desempenho, em termos de distância média, e a Tabela 2.7 mostra que esta combinação é superior, tanto em termos de distância como também, em número de vitórias. Nenhuma combinação, em particular, teve um desempenho ruim nas três versões do *scatter search* em termos de distância média ou número de vitórias, entretanto, a combinação D apresentou o pior desempenho em termos de distância máxima, para as três versões do *scatter search*.

Tabela 2.6 Desempenho da versão do *scatter search* (5, 5)/ *QD*/ *Estático*) com relação aos tipos de combinação

	n				**Distância média (%)**	**Distância máx. (%)**	**Num. vitórias**
	30	**45**	**60**	**75**			
Comb. A	1,29	0,73	1,36	1,07	1,11	4,02	6
Comb. B	1,11	0,65	1,15	1,41	1,08	5,20	10
Comb. C	2,07	0,90	1,49	1,11	1,39	9,18	6
Comb. D	1,55	0,70	1,05	1,85	1,29	9,62	9
Comb. E	1,39	1,17	1,18	0,83	1,14	8,50	9
Comb. F	1,31	0,96	0,88	1,04	1,05	5,20	7
Comb. G	0,46	0,67	0,79	0,85	0,69	1,99	10
Comb. H	0,88	0,52	0,85	0,80	0,76	5,47	13

Tabela 2.7 Desempenho da versão do *scatter search* (7, 3)/ *QD*/ *Dinâmico*) em relação aos métodos de combinação

	n				**Distância média (%)**	**Distância máx. (%)**	**Num. vitórias**
	30	**45**	**60**	**75**			
Comb. A	1,39	0,48	1,36	0,86	1,02	5,04	11
Comb. B	0,89	0,81	1,21	1,08	1,00	3,67	7
Comb. C	0,75	0,90	1,01	0,77	0,86	2,96	9
Comb. D	1,68	0,61	0,98	1,08	1,09	5,20	10
Comb. E	1,55	0,90	1,15	1,16	1,19	4,02	6
Comb. F	0,96	0,78	1,08	0,97	0,95	3,78	10
Comb. G	0,55	0,73	0,73	0,52	0,63	4,02	15
Comb. H	1,31	0,93	1,19	1,08	1,12	4,02	6

Tabela 2.8 Desempenho da versão do *scatter search* (5, 5)/ *Q*/ *Dinâmico*) em relação aos métodos de combinação

	n				**Distância média (%)**	**Distância máx. (%)**	**Num. vitórias**
	30	**45**	**60**	**75**			
Comb. A	1,29	0,77	1,28	0,91	1,06	4,02	9
Comb. B	1,05	0,97	1,06	1,27	1,09	4,20	6
Comb. C	0,69	0,65	1,31	0,71	0,84	2,79	10
Comb. D	1,07	0,51	0,80	0,92	0,82	5,20	13
Comb. E	0,78	1,00	1,06	0,92	0,94	4,02	9
Comb. F	1,53	0,93	1,00	0,62	1,02	5,20	10
Comb. G	1,10	0,52	0,54	0,57	0,68	3,78	11
Comb. H	1,78	0,42	0,96	0,93	1,02	4,34	7

A Tabela 2.9 mostra uma comparação do *scatter search* utilizando o método de geração de soluções diversas com matriz de freqüência, como descrito no Capítulo 1, ou gerando as soluções diversas de forma aleatória. Para este teste, foi utilizada a versão do *scatter search* (5,5)/ *QD*/ *Estático*, com combinação tipo G. É possível observar que o *scatter search* com diversificação por freqüência apresenta melhores resultados do que uma diversificação totalmente aleatória.

Tabela 2.9 Desempenho do *scatter search* utilizando diversificação por freqüência ou aleatória

n	**Diversificação por freqüência**	**Diversificação aleatória**
30	0,46	1,04
45	0,67	0,94
60	0,79	1,09
75	0,85	1,01
Distância média (%)	0,69	1,02
Distância máx. (%)	1,99	4,02
Num. vitórias	10	7

A Tabela 2.10 refere-se à capacidade da heurística de melhoria em conseguir tornar soluções infactíveis em soluções factíveis. Para este experimento, foi utilizada a versão do *scatter search* com parâmetros (5,5)/ *QD*/ *Estático*, com combinação tipo G. A Tabela 2.10 mostra a porcentagem de soluções infactíveis que foram viabilizadas na primeira tentativa de viabilização da Fase 1 (linha "fact.1" da Tabela 2.10) ou segunda tentativa de viabilização da Fase 1 (linha "fact. 2" da Tabela 2.10), e a porcentagem de soluções que não puderam ser viabilizadas. Note que, em média, a Fase 1 é capaz de viabilizar mais de 90% das soluções. Uma possível modificação da Fase 1 seria aplicar até *n* tentativas de viabilização, ao invés de apenas duas. Esta modificação foi testada utilizando a versão do *scatter search* (5,5)/ *QD*/ *Estático*, com combinação tipo G. Praticamente todas as soluções puderam ser viabilizadas, mas os resultados obtidos com esta mudança foram de pior qualidade do que os resultados gerados pela versão com duas tentativas de viabilização, como pode ser visto na Tabela 2.11.

Tabela 2.10 Porcentagem de soluções que foram viabilizadas pela heurística de melhoria

	n				**Média**
	30	**45**	**60**	**75**	
fact. 1 (%)	85,52	86,13	86,60	79,67	84,48
fact. 2 (%)	7,51	10,15	8,47	14,51	10,16
infact. (%)	6,97	3,72	4,93	5,82	5,36

Tabela 2.11 *Scatter search* com *n* tentativas de viabilização

	n				**Distância média (%)**	**Distância máx. (%)**	**Num. vitórias**
	30	**45**	**60**	**75**			
2- tentativas de viabil.	0,46	0,67	0,79	0,85	0,69	1,99	10
***n*-tentativas de viabil.**	1,08	0,53	0,62	0,85	0,77	4,02	9

O próximo experimento também refere-se à fase de viabilização da heurística de melhoria, e os parâmetros do *scatter search* são os mesmos do experimento anterior. Ao invés da estratégia proposta no Capítulo 1, que consiste em selecionar as atividades mais atrasadas, e aumentar os recursos de acordo com as necessidades destas atividades, neste experimento as atividades que determinam o aumento da disponibilidade de recursos são selecionadas

aleatoriamente. Da mesma forma que no Capítulo 1, no máximo duas tentativas de viabilização são realizadas. Para cada valor de *n*, a Tabela 2.12 mostra a porcentagem de soluções que foram viabilizadas, com este novo critério (SSCA – *scatter search* com critério aleatório), considerando 10 problemas testes. Observe que, em média, SSCA viabilizou um número menor de soluções do que o método baseado em atraso (Tabela 2.10). A Tabela 2.13 mostra a distância média, para cada valor de *n*, obtida por SSCA e pelo método baseado em atraso, considerando 10 problemas testes. É possível observar que a versão SSCA obteve mais vitórias do que o método baseado em atraso, porém seu desempenho não foi tão bom com respeito às distâncias média e máxima, como mostram os resultados da Tabela 2.13.

Tabela 2.12 Porcentagem de soluções que foram viabilizadas

	n				**Média**
	30	**45**	**60**	**75**	
Fact. 1 (%)	74,36	78,91	79,13	79,11	77,88
Fact. 2 (%)	15,44	13,24	12,73	12,62	13,51
Infact. (%)	10,20	7,86	8,14	8,27	8,62

Tabela 2.13 Desempenho do *scatter search* com respeito a SSCA

	n				**Distância média (%)**	**Distância máx. (%)**	**Num. vitórias**
	30	**45**	**60**	**75**			
Fact. utilizando atraso	0,46	0,67	0,79	0,85	0,69	1,99	10
SSCA	0,33	0,68	1,04	1,01	0,77	2,16	12

2.4 Resultados computacionais para o Conjunto 2

O objetivo desta seção é avaliar o desempenho da melhor versão estática e das duas melhores versões dinâmicas do *scatter search*, identificadas na Seção 2.3, isto é, versões SS1: (5, 5)/ *QD*/ *Estático*, SS2: (7, 3)/ *QD*/ *Dinâmico*, SS3: (5,5)/ *Q*/ *Dinâmico*. Estas versões são testadas num conjunto de problemas testes envolvendo 30, 60, 90, e 120 atividades e 4, 6, e 8 recursos, com os seguintes parâmetros:

- Fator de recursos (*RF*): 0,25, 0,5, 0,75 e 1,0.
- Complexidade de rede (*NC*): 1,5, 1,8 e 2,1.
- Fator de data de entrega (*DF*): 1,0, 1,2 e 1,4.

Para cada combinação de *RF, DF, NC, n* e *m*, um problema teste foi gerado, resultando num total de $4 \cdot 3 \cdot 3 \cdot 4 \cdot 3 = 432$ problemas testes. Para SS1, SS2 e SS3, utilizou-se a combinação do tipo G, *Psize* = 30, *g* = 10 e λ = 1,9. O critério de parada para SS1 foi *MaxIter* = 5 iterações, com limite de tempo de uma hora. O número de soluções avaliadas para esta versão foi adotado como critério de parada para SS2 e SS3. A Tabela 2.14 mostra o número médio de soluções avaliadas, para 36 problemas testes.

Tabela 2.14 Número médio de soluções avaliadas para o Conjunto 2.

m	***n***			
	30	**60**	**90**	**120**
4	702253	1109530	1319468	1013355
6	956848	1422224	1453986	961312
8	1307508	1580518	1391682	873373

Tabela 2.15 Desempenho de SS1 de acordo com fator de recurso e número de atividades

RF	*n*				Distância média (%)	Num. vitórias	Distância máx. (%)
	30	60	90	120			
0,25	0,01	0,27	0,55	0,47	0,32	68	0,04
0,50	0,19	0,28	0,45	0,38	0,33	45	3,14
0,75	0,17	0,45	0,28	0,63	0,38	42	2,53
1,00	0,43	0,40	0,18	0,53	0,39	44	3,81
Média (%)	0,20	0,35	0,37	0,50	0,35		
Total						199	

Tabela 2.16 Desempenho de SS2 de acordo com fator de recurso e número de atividades

RF	*n*				Distância média (%)	Num. vitórias	Distância máx. (%)
	30	60	90	120			
0,25	0,01	0,14	0,32	0,43	0,22	64	0,04
0,50	0,22	0,46	0,25	0,33	0,31	53	2,01
0,75	0,10	0,61	0,49	0,48	0,42	40	2,48
1,00	0,44	0,27	0,44	0,21	0,34	43	3,03
Média (%)	0,19	0,37	0,37	0,36	0,32		
Total						200	

Tabela 2.17 Desempenho de SS3 de acordo com fator de recurso e número de atividades

RF	*n*				Distância média (%)	Num. vitórias	Distância máx. (%)
	30	60	90	120			
0,25	0,01	0,04	0,54	0,23	0,21	68	0,04
0,50	0,16	0,33	0,32	0,57	0,35	55	2,53
0,75	0,51	0,21	0,43	0,17	0,33	54	3,78
1,00	0,26	0,48	0,56	0,24	0,39	39	3,35
Média (%)	0,23	0,27	0,46	0,30	0,32		
Total						216	

Para cada valor de número de atividade *n* e fator de recurso *RF*, as Tabelas 2.15, 2.16 e 2.17 mostram a distância média, em porcentagem, sobre 27 problemas testes. Os números nas últimas três colunas são calculados com respeito a 108 problemas testes, e é possível observar que, em geral, SS2 e SS3 têm um desempenho um pouco melhor do que SS1. Dentre as três versões, SS1 apresenta o maior valor de distância máxima. É interessante notar também que, nas três versões, a maior distância máxima ocorre em problemas testes com $RF = 0{,}75$ e $RF = 1{,}0$. Comparando SS2 com SS3, pode-se observar que SS2 apresenta um número de vitórias menor do que SS3, entretanto, SS3 também apresenta valores de distância máxima maiores ou iguais a SS2 para todos os valores de *RF*.

Para cada valor de *n* e *m*, a Tabela 2.18 exibe a distância média, em porcentagem, para 36 problemas testes. Os números nas últimas três colunas são calculados com respeito a 144 problemas testes. Os dados nesta tabela mostram que não há uma versão dominante com relação ao número de recursos. SS1 apresenta distância média menor e número de vitórias maior, do que SS2 e SS3 para $m = 4$. Em termos de distância máxima, note que os maiores valores de distância máxima, para SS1, SS2 e SS3, ocorrem em $m = 4$.

A influência da complexidade de rede *NC* é exibida na Tabela 2.19. É interessante observar que a distância média decresce à medida que *NC* aumenta, para SS1, SS2 e SS3.

A Tabela 2.20 mostra a influência do fator de data de entrega no desempenho de SS1, SS2 e SS3. Para todo valor de *DF*, SS2 e SS3 apresentam uma distância média menor do que SS1. A última coluna da Tabela 2.20 mostra o valor médio da função objetivo, de acordo com os valores de *DF*. Drexl e Kimms (2001) observam que a resolução do PCDR com diversas datas de entrega dá uma idéia do balanço tempo/custo, entre terminar um projeto rapidamente e realizar o projeto de forma mais econômica. Isto pode ser uma informação valiosa ao negociar o preço do projeto.

Tabela 2.18 Desempenho de SS1, SS2 e SS3 com respeito ao número de recursos e número de atividades

	m	*n*				Distância média (%)	Num. vitórias	Distância máx. (%)
		30	**60**	**90**	**120**			
SS1	**4**	0,29	0,17	0,41	0,36	0,31	81	3,81
	6	0,38	0,46	0,24	0,67	0,44	67	3,48
	8	0,38	0,42	0,45	0,48	0,43	51	3,14
SS2	**4**	0,25	0,38	0,46	0,32	0,35	69	3,03
	6	0,09	0,23	0,32	0,38	0,26	68	2,35
	8	0,23	0,49	0,34	0,39	0,36	63	2,25
SS3	**4**	0,36	0,26	0,49	0,33	0,36	78	3,78
	6	0,26	0,30	0,44	0,20	0,30	75	3,61
	8	0,33	0,23	0,46	0,38	0,35	63	3,18

Tabela 2.19 Desempenho de SS1, SS2 e SS3 com respeito à complexidade de rede

	NC	Distância média (%)	Num. vitórias	Distância máx. (%)
SS1	**1,5**	0,43	70	3,48
	1,8	0,32	69	3,81
	2,1	0,30	60	2,11
SS2	**1,5**	0,35	54	2,79
	1,8	0,34	66	2,48
	2,1	0,28	80	3,03
SS3	**1,5**	0,34	71	3,78
	1,8	0,30	75	2,23
	2,1	0,31	70	3,35

Tabela 2.20 Desempenho de SS1, SS2 e SS3 com respeito ao fator de data de entrega

	DF	Distância média (%)	Num. vitórias	Distância máx. (%)	Custo médio
SS1	**1,0**	0,31	76	3,01	822,11
	1,2	0,41	65	3,81	657,39
	1,4	0,45	58	3,14	566,12
SS2	**1,0**	0,26	72	3,03	821,98
	1,2	0,40	61	2,48	657,46
	1,4	0,32	67	2,79	565,31
SS3	**1,0**	0,28	73	3,35	821,94
	1,2	0,34	69	3,78	656,90
	1,4	0,39	74	3,18	565,81

Um experimento interessante é verificar a influência dos tipos de subconjuntos (ST) em SS1, SS2 e SS3. Isto pode ser feito ao contar o número de vezes que a melhor solução foi encontrada pelos métodos de combinação e geração de soluções diversas. As Figuras 2.1, 2.2 e 2.3 mostram o número de vezes em que a melhor solução de SS1, SS2 e SS3, respectivamente, foi encontrada pelo método de geração de soluções diversas, subconjunto do tipo 1, subconjunto do tipo 2, subconjunto do tipo 3, e subconjunto do tipo 4. Para um número significativo de problemas testes com 30 atividades, a melhor solução é encontrada pelo método de geração de soluções diversas. Entretanto, à medida que o número de atividades aumenta, na maior parte dos problemas testes, a melhor solução é encontrada pelo método de combinação. Note que a distribuição das combinações é mais uniforme em SS1 do que em SS2 e SS3. Isto ocorre devido à atualização dinâmica do conjunto de referência em SS2 e SS3 que, ao contrário da atualização estática, pode não realizar todas as combinações de elementos de *RefSet*.

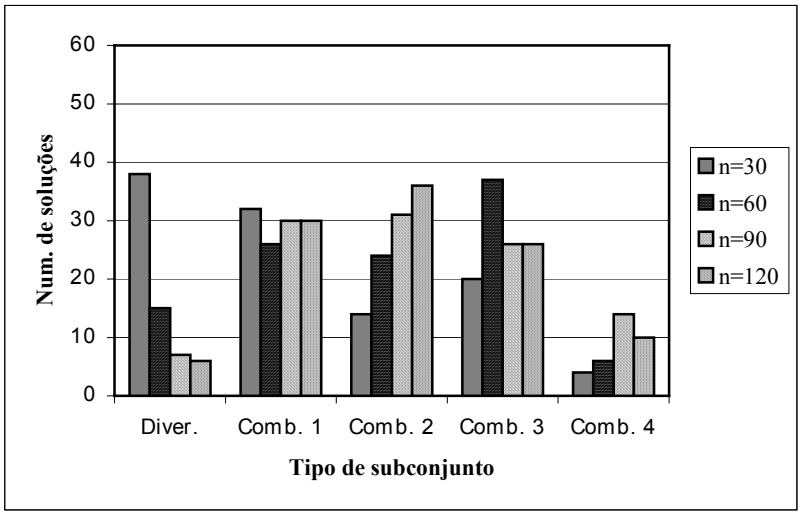

Figura 2.1 Histograma de soluções para SS1

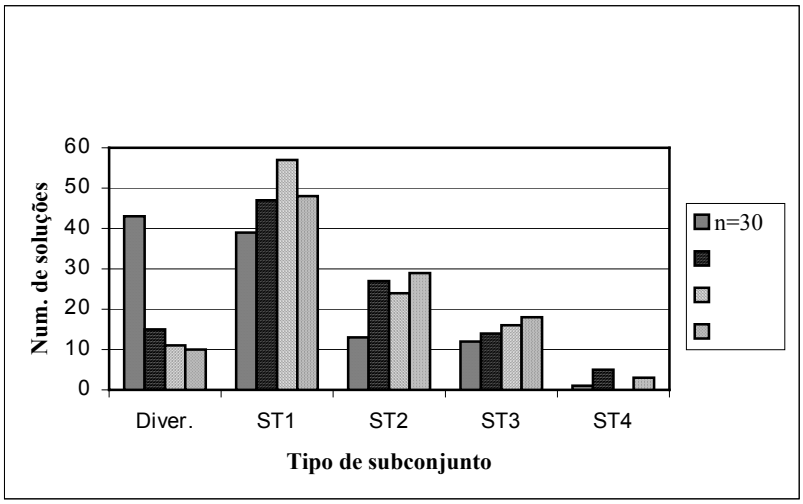

Figura 2.2 Histograma das soluções para SS2

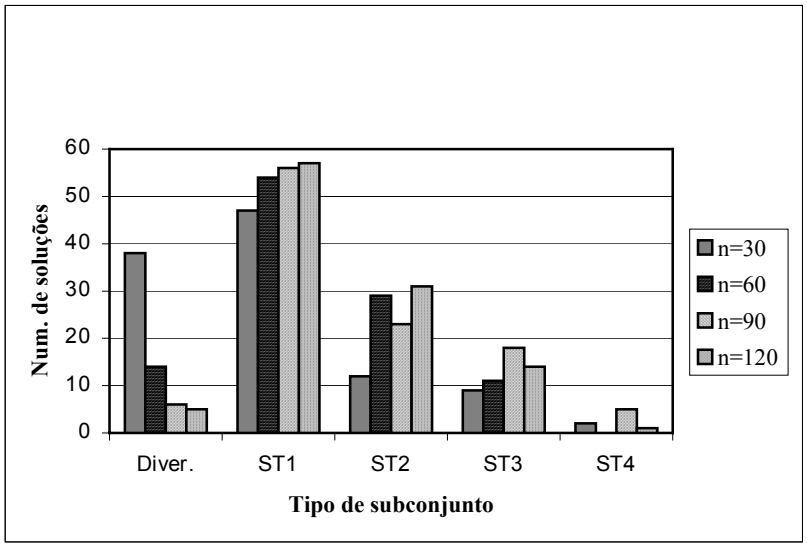

Figura 2.3 Histograma das soluções para SS3

Tabela 2.21 Desempenho de SS1 e SS2 ao utilizar os 4 tipos de subconjunto ou utilizando ST1 somente.

	n				**Distância**	**Número**	**Distância**
	30	**60**	**90**	**120**	**média (%)**	**de vitórias**	**máx. (%)**
SS1	0,38	0,22	0,28	0,65	0,38	24	3,81
SS1 (ST1 apenas)	0,90	0,68	0,93	1,25	0,94	11	3,78
SS2	0,46	0,44	0,30	0,46	0,42	22	2,48
SS2 (ST1 apenas)	0,66	0,32	0,65	0,80	0,61	16	3,92
SS3	0,45	0,28	0,28	0,50	0,38	23	3,78
SS3 (ST1 apenas)	0,61	0,53	0,94	0,47	0,63	16	2,79

O objetivo do próximo experimento é analisar o desempenho de SS1, SS2 e SS3 utilizando somente ST1, ao invés de todos os tipos de subconjuntos. Os experimentos computacionais foram limitados a problemas testes do Conjunto 2 com $m = 4$ e $DF = 1{,}2$. Observe que a qualidade das soluções das três versões, principalmente SS1, deteriora ao utilizar apenas ST1.

A Tabela 2.22 mostra a média dos tempos computacionais, em segundos, gasto pelas heurísticas SS1, SS2 e SS3. Note que, se o tempo limite for atingido durante uma combinação de soluções, então o programa completa esta combinação antes de parar. Se isto acontecer, o limite de tempo de 3600 segundos pode ser excedido, como mostrado na Tabela 2.22.

Tabela 2.22 Média do tempo computacional gasto pelas versões do *scatter search*

m	***n***			
	30	**60**	**90**	**120**
4	369,12	1454,63	3273,03	3698,17
6	557,00	2086,97	3690,12	3741,33
8	851,10	2435,84	3700,70	3871,42

2.5 Resultados computacionais para o Conjunto 3

Nesta seção, o desempenho da versão *scatter search* (5,5)/ *QD*/ *Estático*, com combinação G, é comparado a limitantes superiores, limitantes inferiores e soluções exatas. O restante dos parâmetros desta versão são *MaxIter* = 5, *PSize* = 30, λ = 1,9, g = 10. Soluções ótimas são geradas por um algoritmo de planos de corte proposto por Demeulemeester (1995), com um limite de tempo de 600 segundos. Se uma solução ótima não puder ser encontrada dentro deste limite de tempo, então a solução infactível resultante pode gerar um limitante inferior para o problema. O algoritmo proposto por Demeulemeester (1995) é descrito no Apêndice B deste trabalho. Limitantes superiores e inferiores para o PCDR são obtidos por uma relaxação lagrangiana e por um método de geração de colunas proposto por Drexl e Kimms (2001). Os autores sugerem conjuntos de problemas testes para 30 e 20 atividades, como descrito abaixo.

Conjunto de 30-atividades: Consiste de problemas testes com 4 recursos, obtidos da biblioteca PSPLIB e adaptados para o PCDR. Os seguintes parâmetros foram utilizados:

- Fator de recurso (*RF*): 0,25, 0,5, 0,75 e 1,0
- Complexidade de rede (*NC*): 1,5, 1,8 e 2,1
- Fator de data de entrega (*DF*): 1,0, 1,1, 1,2, 1,3, 1,4 e 1,5

Para cada combinação de *RF*, *NC* e *DF*, 40 problemas testes foram gerados, totalizando $4 \times 3 \times 6 \times 40 = 2880$ problemas testes. 1249 problemas testes que excederam 500 colunas durante a execução do método de geração de colunas foram descartados por Drexl e Kimms (2001). O procedimento exato de Demeulemeester (1995) forneceu soluções ótimas ou limitantes inferiores para 2050 problemas testes.

Conjunto de 20-atividades: Consiste de problemas testes gerados pelo programa Progen com n = 20 atividades e $m \in \{2,4,6,8\}$ tipos de recursos. Os seguintes parâmetros foram utilizados para gerar os problemas testes:

- Fator de recurso (*RF*): 1,0
- Complexidade de rede (*NC*): 1,5
- Fator de data de entrega (*DF*): 1,0, 1,1, 1,2, 1,3, 1,4 e 1,5

Para cada combinação de *m, RF*, *NC* e *DF*, 10 problemas testes foram gerados, totalizando $4 \times 6 \times 10 = 240$ problemas testes. 7 problemas testes que excederam 2500 colunas durante a execução do método de geração de colunas foram descartadas por Drexl e Kimms (2001). O procedimento exato de Demeulemeester (1995) forneceu soluções ótimas ou limitantes inferiores para 232 problemas testes.

A Tabela 2.23 mostra a distância média do *scatter search* com respeito à solução exata e ao limitante inferior (LI) para o conjunto com 30 e 20 atividades. A distância média do *scatter search* com respeito à solução ótima foi de 0,3% e 0,78%, para os conjuntos com 30 e 20 atividades, respectivamente. A versão *scatter search* encontrou a solução ótima em aproximadamente 85% dos problemas testes com 30 atividades, para os quais a solução ótima está disponível, e 71% para o conjunto de 20-atividades. Para problemas testes em que a solução ótima não está disponível, a versão *scatter search* é comparada com o melhor limitante inferior

gerado por Demeulemeester (1995) e Drexl e Kimms (2001). As últimas duas colunas da tabela mostram a distância média dos limitantes superiores (LS) encontrados por Drexl e Kimms (2001), com respeito à solução ótima e limitantes inferiores.

Tabela 2.23 Desempenho do *scatter search* para o conjunto de 30-atividades e 20-atividades

	Distância média de SS para ótimo (%)	**Número de soluções ótimas de SS**	**Número de soluções ótimas disponíveis**	**Distância média de SS para LI (%)**	**Número de limitantes inferiores**	**Distância média de LS para ótimo (%)**	**Distância média de LS para LI (%)**
30 - atividades	0,30	1199	1403	7,65	791	26,58	36,98
20 - atividades	0,78	75	105	8,27	134	22,73	26,18

A Tabela 2.24 mostra o tempo computacional médio gasto, em segundos, pelo método de geração de colunas, relaxação lagrangiana, *scatter search* e algoritmo de planos de corte. É interessante notar que o tempo total gasto pelo *scatter search* para encontrar a solução ótima é maior do que o tempo gasto pelo algoritmo exato, entretanto, o tempo gasto pelo *scatter search* para encontrar a melhor solução é sempre menor do que o tempo gasto pelo algoritmo exato. O método de geração de colunas e a relaxação lagrangiana foram executados num Pentium II, 300 MHz, e 64 MB RAM (Drexl e Kimms, 2001).

Tabela 2.24 Tempo computacional médio, em segundos, para os conjuntos com 30 atividades e 20 atividades

	Geração de colunas	**Relaxação lagrangiana**	**Algoritmo exato**	**SS tempo melhor sol.**	**SS tempo total**
30-atividades	15,09	2,10	162,69	61,61	296,75
20-atividades	889,30	0,94	162,46	41,40	234,64

A Tabela 2.25 mostra o número relativo de problemas testes, cuja distância em relação à solução ótima, está contida dentro de um certo intervalo. Note que, para a maioria dos problemas testes, o valor da distância em relação à solução ótima, é menor do que 1%, e em nenhum problema teste, a distância é maior do que 10%.

Tabela 2.25 Distribuição dos problemas testes, de acordo com distâncias, para os conjuntos de 30-atividades e 20-atividades

	Intervalo de distância (%)				
	[0,1)	**[1,3)**	**[3,5)**	**[5,10)**	**[10,...)**
30 -atividades	90,24	6,99	2,00	0,78	0,00
20 -atividades	80,00	9,52	7,62	1,90	0,00

2.6 Conclusões

No Capítulo 1 deste trabalho foi proposto um método de resolução, baseado na heurística *scatter search*, para o problema de programação de projetos conhecido como Problema de Custo de Disponibilidade de Recursos. Diversos métodos de combinação de solução e estratégias avançadas para atualizar o conjunto de referência foram propostos. O desempenho das melhores versões foi testado em um grande número de problemas. Para problemas de pequeno porte, o desempenho de tais versões foi comparado a soluções exatas, limitantes inferiores e superiores da literatura. Para problemas de médio porte, desenvolveram-se heurísticas de múltiplos inícios simples com a finalidade de mostrar o ganho de qualidade gerado por uma heurística mais complexa, como *scatter search*. O *scatter search* foi capaz de encontrar soluções ótimas ou próximas do ótimo, num número significativo de problemas testes, e obteve melhores resultados do que as heurísticas de múltiplos inícios em 92,5% dos problemas testes. Em geral, o método *scatter search* é robusto, uma vez que seu desempenho não é muito sensível com relação às versões testadas. Os resultados deste trabalho mostram que o método proposto é capaz de resolver PCDRs de grande porte, e gerar soluções de boa qualidade, num tempo computacional razoável.

CAPÍTULO 3

O PROBLEMA DE CUSTO DE DISPONIBILIDADE DE RECURSOS COM CENÁRIOS

Este capítulo aborda o problema de disponibilidade de recursos com incerteza em relação aos tempos de processamento das atividades. São apresentados modelos de minimização do desvio máximo (*maximum regret*) e média-variância, que são resolvidos através do método *scatter search* proposto no Capítulo 1 deste trabalho.

3.1 Descrição do problema

Muitos problemas de decisão, representados por modelos matemáticos, contêm algum grau de incerteza em seus dados. Este capítulo aborda o problema de programação de projetos PCDR, apresentado no Capítulo 1, com incerteza nas durações das atividades. Neste problema, o decisor deseja determinar a quantidade de recursos que deve ser alocada para o projeto, sem saber *a priori*, a duração das atividades. Enquanto muitos estudos envolvendo incerteza otimizam um resultado médio, o objetivo deste trabalho é utilizar uma abordagem para controlar a variabilidade da solução. Nesta abordagem, chamada de otimização robusta (Mulvey *et al*., 1995), as durações das atividades são representadas como um conjunto de possíveis cenários e procura-se uma solução que seja robusta em relação aos cenários.

Otimização robusta tem sido aplicada a problemas de diversas áreas como expansão de capacidade energética (Malcolm e Zenios, 1994), planejamento de expansão de capacidade de uma facilidade em redes de telecomunicações (Laguna, 1998), planejamento da produção

(Escudero *et al*., 1999), e programação da produção (Daniels e Kouvelis, 1995). A escolha de um critério de decisão adequado deve refletir a atitude do decisor com respeito à incerteza. Neste trabalho, são considerados dois modelos para o PCDRC (problema de custo de disponibilidade de recursos com cenários) que diferem pela função objetivo. No primeiro deles, deseja-se encontrar uma decisão que minimize o desvio máximo *(maximum regret*), isto é, o maior desvio da otimalidade dentre todas as decisões, sobre todos os possíveis cenários, enquanto o segundo trata da minimização da função média-variância. Uma heurística de melhoria é proposta para cada um destes modelos, que são então resolvidos através de uma versão do método *scatter search* proposto no Capítulo 1.

Modelo para minimização do desvio máximo

Considere um projeto com n atividades sujeito a relações de precedência $(i , j) \in H$ e com atividades fictícias 1 e n que representam o início e o fim do projeto. Cada atividade requer r_{ik} unidades de recurso do tipo k, $k = 1,...,m$ durante sua execução e tem duração d_{is}, tal que s é um cenário de um conjunto finito S. O projeto tem um limite de tempo representado por uma data de entrega D, e uma função de custo não decrescente $C_k(a_k)$ associada à disponibilidade de recursos a_k. As variáveis de decisão do problema são o instante de término da atividade i, representado por f_i, e a disponibilidade a_k de recursos do tipo k . Sejam $K = (a_1, ..., a_m)$ e $d_s = (d_{1s},...,d_{ns})$, o vetor de disponibilidades de recurso e o vetor de duração das atividades para um dado cenário s, respectivamente. Um modelo conceitual do problema de minimização do desvio máximo para o PCDR com cenários (PCDRC-DevMax) (Rangaswamy, 1998) é apresentado a seguir.

$$\underset{K}{Minimizar} \qquad \Psi(K) \tag{1}$$

tal que

$$\Psi(K) = \max_{s \in S}(\Gamma(K, s))$$

e

$$\Gamma(K, s) = \left[\sum_k \left(C_k\left(a_k\right) - C_k\left(\delta_k^s\right)\right)\right] + \rho \cdot \max\left(0, \Phi(d_s, K) - D\right)$$

O custo $\Gamma(K,s)$, para um vetor de disponibilidade de recursos *K* e um cenário *s*, é composto por dois termos. O primeiro representa o desvio de custo em relação à δ_k^s, que corresponde à disponibilidade ótima do recurso *k* para o cenário *s*. Para cada cenário, a solução ótima $\delta^s = (\delta_1^s,...,\delta_m^s)$ é obtida ao se resolver o PCDR descrito no Capítulo 1 deste trabalho. O segundo termo é uma penalidade por completar o projeto depois de sua data de entrega *D*. O parâmetro ρ é um fator de penalização dado, e $\Phi(d_s,K)$ representa o *makespan* ótimo do projeto para uma dada disponibilidade de recursos *K* e um dado vetor de duração d_s. O valor de $\Phi(d_s,K)$ é obtido através da solução do problema de programação de projetos com recursos limitados (PPPRL).

Modelo para minimização de média-variância

O segundo modelo aqui tratado, considera a média e a variância dos custos. Como no modelo anterior, as variáveis de decisão deste problema são o instante de término da atividade *i*, representado por f_i, e a disponibilidade a_k de recursos do tipo *k*. Um modelo conceitual do problema de minimização da média-variância para o PCDR com cenários (PCDRC-MVar) (Rangaswamy, 1998) é apresentado a seguir.

$$\underset{K}{Minimizar} \qquad MV(K) = MD(K) + \beta \cdot \sum_{s \in S} p_s \cdot (MD(K) - CP(K,s))^2 \qquad (2)$$

tal que

$$CP(K,s) = \sum_k C_k(a_k) + \rho \cdot \max(0, \Phi(d_s,K) - D) \qquad \text{e}$$

$$MD(K) = \sum_{s \in S} p_s \cdot CP(K,s)$$

O modelo acima tem como objetivo encontrar a disponibilidade de recursos *K* que minimiza a média e a variância dos custos relativos a todos os cenários considerados. O parâmetro β, em (2), é um fator de variância determinado pelo decisor, e quanto maior o seu

valor, maior é a aversão do decisor a risco. Para uma dada disponibilidade de recursos *K, CP*(*K,s*) é o valor da função de custo do PCDR penalizada, para o cenário *s*. O primeiro termo de *CP*(*K,s*) representa o custo do projeto, e o segundo termo é uma penalidade por completar o projeto depois de sua data de entrega *D*. O parâmetro ρ é um fator de penalização dado, e $\Phi(d_s, K)$ representa o *makespan* ótimo do projeto para uma dada disponibilidade de recursos *K* e um dado vetor de duração d_s. Como observado no modelo anterior, o valor de $\Phi(d_s, K)$ é obtido através da solução do problema de programação de projetos com recursos limitados (PPPRL). Finalmente, *MD*(*K*) refere-se à média dos valores de *CP*(*K,s*), e p_s é a probabilidade do cenário *s* ocorrer. Neste trabalho, os cenários têm a mesma probabilidade de ocorrer, ou seja, $p_s = 1/|S|$, $s = 1,...,|S|$.

Deve-se notar que os dois modelos apresentados acima são modelos de dois-estágios com recurso (*recourse*). Recurso pode ser definido como a possibilidade de tomar uma decisão, ou corrigir uma ação, após a incerteza ter sido resolvida. Nos modelos estudados neste capítulo, considera-se que no primeiro estágio, o decisor precisa determinar a disponibilidade de recursos sem saber *a priori* as durações das atividades. No segundo estágio, após a incerteza ter sido eliminada, o decisor tem a possibilidade de programar novamente as atividades, considerando a disponibilidade de recursos encontrada no primeiro estágio.

O objetivo deste capítulo é aplicar o método populacional *scatter search* aos modelos de otimização robusta descritos acima. Os problemas PCDR e PPPRL são resolvidos utilizando os métodos heurísticos *scatter search* e ASP, descritos no Capítulo 1.

3.2 Otimização robusta

A técnica de otimização robusta utilizada neste trabalho foi introduzida por Mulvey *et al*. (1995) e tem como objetivo encontrar uma solução que seja robusta em relação aos cenários considerados. Mulvey *et al*. (1995) sugerem que este objetivo pode ser atingido abordando dois aspectos do problema. O primeiro aspecto refere-se às restrições de factibilidade. Nem sempre é possível encontrar uma solução que seja factível para todos os cenários, e portanto, Mulvey *et al*. (1995) propõem a utilização de funções de penalidade, isto é, algumas restrições do problema podem ser relaxadas e adicionadas à função objetivo com um fator que penaliza a infactibilidade da solução. O segundo aspecto refere-se à escolha adequada da função objetivo do problema. Em

geral, as técnicas tradicionais de programação estocástica procuram otimizar um resultado médio, o que não é apropriado para situações que envolvam risco ou incerteza. Nestes casos, Mulvey *et al*. (1995) sugerem a utilização de outras funções objetivo que reflitam melhor a noção de risco do decisor, como a minimização de média-variância (Markowitz, 1959), minimização do desvio máximo (*maximum regret*), funções de utilidade, entre outras.

3.3 Revisão bibliográfica

A revisão bibliográfica descrita a seguir está dividida em duas seções: na Seção 3.3.1 são apresentadas algumas aplicações que utilizam otimização robusta, e na Seção 3.3.2 são descritos trabalhos de programação de projetos com incerteza, resolvidos através de métodos de programação estocástica. De nosso conhecimento, não existem trabalhos na literatura, que abordam o problema de custo de disponibilidade de recursos com incerteza.

3.3.1 Otimização robusta

A técnica de otimização robusta tem sido aplicada a problemas de diversas áreas, como por exemplo em Escudero *et al*. (1999), Daniels e Kouvelis (1995), Gutierrez e Kouvelis (1995), Yu e Li (2000), Malcolm e Anandalingam (2000), Escudero *et al*. (1993), Laguna (1998).

Os seguintes trabalhos minimizam o custo esperado com penalização de uma ou mais restrições violadas. Valls *et al.* (1998) abordam o problema de programação de projetos com recursos limitados, no qual algumas das atividades podem ter seu processamento interrompido por um período de tempo incerto. Cada atividade tem uma data de entrega e se a atividade terminar após sua data de entrega, uma penalidade é somada ao atraso da atividade. O objetivo é encontrar uma solução que minimize o atraso ponderado total esperado. O problema é resolvido pelo método *scatter search*, e resultados computacionais são relatados para 162 problemas testes com 50, 100 e 150 atividades e 4 recursos, considerando 30, 50 e 70 cenários. Os autores mostram que, ao aplicar a solução obtida considerando 50 cenários nos problemas com 70 cenários, foi possível gerar resultados melhores do que os obtidos ao resolver o problema considerando 70 cenários. Isto indica que o procedimento é robusto, uma vez que boas soluções

podem ser encontradas com um número de cenários relativamente pequeno. Outro trabalho que também tem como objetivo minimizar o custo esperado é o trabalho de Escudero *et al*. (1999). O artigo apresenta um modelo de otimização de um problema de programação de suprimento, transformação e distribuição multiperíodos, com incerteza na demanda do produto, custo de suprimento à vista e preço de venda à vista. Os autores sugerem métodos de solução baseados em decomposição de Benders e lagrangiano aumentado, mas resultados computacionais não são apresentados. Laguna (1998) aborda o problema de expansão de capacidade de uma única facilidade no planejamento de rede de telecomunicações, com demanda incerta. O objetivo é determinar o menor custo esperado de expansão, penalizando cenários que são infactíveis com respeito à demanda. Um método de solução exata, com duas fases, é proposto para solucionar o problema. A primeira fase é baseada em programação dinâmica e a segunda fase utiliza um procedimento de caminho mínimo.

Aplicações de otimização robusta que minimizam o desvio máximo podem ser encontradas em Gutierrez e Kouvelis (1995) e Daniels e Kouvelis (1995). Gutierrez e Kouvelis (1995) abordam o problema de selecionar uma rede internacional de fornecedores para atender demandas de matéria prima de um grande fabricante. O objetivo é encontrar uma rede de fornecedores que seja insensível (ou robusta) com respeito a possíveis incertezas, ou variações nas taxas de câmbio dos países de origem. A função objetivo utilizada é a minimização do desvio máximo. O problema é modelado como uma variante do problema de localização de facilidades sem capacidade, e um algoritmo *branch-and-bound* é proposto para resolvê-lo. Daniels e Kouvelis (1995), abordam o problema de programação de tarefas em uma máquina, em que os tempos de processamento das tarefas são representados por um conjunto de cenários. O objetivo é encontrar um programa que minimize o desvio máximo do tempo de fluxo total. Regras de dominância e um algoritmo *branch-and-bound* são propostos, assim como heurísticas baseadas em condições de otimalidade para problemas de programação robusta de duas tarefas. Testes computacionais com 10, 15 e 20 tarefas são realizados e os resultados mostram que o modelo proposto é capaz de gerar soluções robustas e de boa qualidade.

Outra aplicação de otimização robusta relacionada à minimização do desvio pode ser vista em Yu e Li (2000). Os autores propõem uma reformulação de otimização robusta para o modelo de programação linear proposto por Mulvey *et al*. (1995) para um problema de

minimização do custo de compra de matéria prima, produção e distribuição de produtos. Na abordagem proposta por Yu e Li (2000), o modelo robusto tem um número menor de variáveis quando comparado ao modelo original de otimização robusta tradicional de Mulvey *et al*. (1995). O método é ilustrado através de dois problemas com até 12 cenários, cujo objetivo é minimizar o desvio médio absoluto e a variabilidade da solução em relação aos cenários.

No trabalho de Malcolm e Anandalingam (2000), os autores procuram um equilíbrio entre a minimização do custo esperado e a minimização do desvio máximo. Malcolm e Anandalingam (2000) apresentam um modelo de otimização robusta para planejamento de expansão de capacidade elétrica que é caracterizado por diversas restrições físicas, de recursos e de demandas. A função objetivo a ser minimizada consiste da soma do custo esperado e do desvio ponderado em relação à solução ótima de cada cenário (*regret*). O modelo é aplicado a um problema real na Índia. As soluções geradas pelo modelo robusto são comparadas com um modelo de programação estocástico cujo objetivo é minimizar o custo médio. Resultados computacionais indicam que a otimização robusta é mais apropriada para lidar com incertezas do que o modelo de programação estocástico.

3.3.2 Programação estocástica

Problemas de programação de projetos com durações variáveis para as atividades e recursos limitados podem ser encontrados na literatura. Em geral, nestes trabalhos, as durações das atividades são representadas por uma função de distribuição de probabilidade, e o objetivo é minimizar a duração esperada do projeto.

Tsai e Gemmill (1998) propõem uma busca tabu que pode ser aplicada ao PPPRL com durações estocásticas ou determinísticas. No caso estocástico, as durações são representadas por uma função de distribuição de probabilidade beta. Uma solução para o problema estocástico pode então ser obtida utilizando a duração esperada de cada atividade e aplicando a busca tabu. A solução gerada pela busca tabu é representada por uma seqüência de atividades, indicando em que ordem elas devem ser executadas. A duração esperada do projeto pode ser obtida da seguinte forma: dada a seqüência obtida pela busca tabu, geram-se os tempos de duração para as atividades utilizando a distribuição beta e avalia-se a duração do projeto. Este passo é repetido

várias vezes, e dessa forma, é possível obter um *makespan* esperado para o problema. A busca tabu proposta utiliza uma vizinhança reduzida baseada em troca de duas atividades.

Golenko-Ginzburg e Gonik (1997) propõem uma heurística para abordar o PPPRL, no qual a duração das atividades é dada por uma distribuição de probabilidade. O objetivo é minimizar a duração esperada do projeto (*makespan*). A heurística utiliza o método paralelo para programar as atividades do projeto. Sempre que um conflito ocorre, devido à restrição de recursos, seleciona-se a atividade que maximiza a contribuição total das atividades na duração esperada do projeto. Para cada atividade, sua contribuição é o produto da duração média da atividade e da probabilidade da atividade estar no caminho crítico do projeto. Estes valores de probabilidade são obtidos via simulação. Testes computacionais não são relatados, mas um exemplo numérico é utilizado para ilustrar o método. Num trabalho posterior, Golenko-Ginzburg e Gonik (1998) aplicam uma heurística semelhante para um problema de programação de projetos no qual existe uma relação entre a duração estocástica das atividades e a quantidade de recursos que as atividades precisam para ser processadas.

Enquanto os trabalhos citados acima utilizam métodos heurísticos, Möhring e Stork (1998) propõem um algoritmo *branch-and-bound* para solucionar o PPPRL com durações estocásticas. Testes computacionais são realizados em 480 problemas testes de pequeno porte.

3.4 Exemplo do problema de custo de disponibilidade de recursos com cenários

Considere um exemplo do PCDR com 12 atividades, $m = 3$ tipos de recurso, data de entrega $D = 18$ e $S = 3$ cenários. A função de custo é dada por $\sum_k C_k(a_k) = c_1a_1 + c_2a_2 + c_3a_3$, com $c_1 = 2$, $c_2 = 2$, $c_3 = 2$, e fator de penalidade por atraso $\rho = 10$. A Figura 3.1 mostra as relações de precedência entre as atividades. A Tabela 3.1 mostra a duração das atividades para cada cenário, e a quantidade de recursos que cada atividade requer para ser processada.

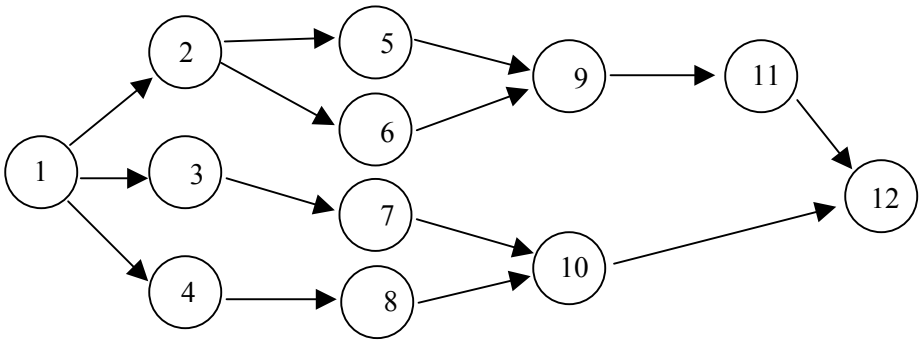

Figura 3.1 Relações de precedência

Tabela 3.1 Dados para o projeto do exemplo.

	Atividades											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
r_{i1}	0	0	2	0	2	0	1	0	2	2	0	0
r_{i2}	0	0	0	1	0	1	1	0	2	0	0	0
r_{i3}	0	1	1	2	0	0	0	1	1	0	2	0
d_1	0	7	3	4	2	5	2	5	4	3	2	0
d_2	0	7	3	4	5	5	2	5	4	3	2	0
d_3	0	2	3	4	7	5	9	7	4	3	2	0

A Tabela 3.2 mostra a solução ótima para cada cenário do exemplo, além dos custos e *makespan* mínimo referente a cada cenário.

Tabela 3.2 Solução ótima para cada cenário

Cenários	**Recursos**			**Custo**	***Makespan***
1	2	2	4	16,00	18
2	4	2	3	18,00	18
3	3	3	2	16,00	17

Suponha que $K = (3,3,4)$ recursos sejam alocados para o problema descrito no exemplo. O cálculo do desvio máximo é ilustrado para este exemplo. Após resolver o PPPRL para cada cenário, obtém-se o *makespan* mínimo, e o valor do desvio pode ser calculado. A Tabela 3.3 mostra os valores do *makespan*, custo, penalidade e desvio para cada cenário. Note que os cenários 1 e 3 são factíveis, ao contrário do cenário 2, que está uma unidade atrasado em relação à data de entrega. O desvio máximo é gerado pelo cenário 2, e é igual a 12.

Tabela 3.3 Valor do desvio para a disponibilidade de recursos $K = (3,3,4)$

Cenários	***Makespan***	$\sum_k \left(C_k(a_k) - C_k(\delta_k^s)\right)$	**Penalidade:** $\rho \cdot \max(0, \Phi(d_s, K) - D)$	**Desvio:** $\Gamma(K, s)$
1	18	20-16 = 4	0	4
2	19	20-18 = 2	1·10 = 10	12
3	17	20-16 = 4	0	4

Tabela 3.4 Valor da variância para a disponibilidade de recursos $K = (3,3,4)$

Cenários	***Makespan***	**Custo**	$(MD(K) - CP(K,s))^2$
1	18	20,00	$(23,33 - 20)^2 = 11,09$
2	19	30,00	$(23,33 - 30)^2 = 44,49$
3	17	20,00	$(23,33 - 20)^2 = 11,09$

O cálculo da média-variância é ilustrado na Tabela 3.4. A média da função de custo penalizada, sobre todas as realizações de cenário, é igual a 23,33. A Tabela 3.4 mostra os valores de *makespan*, função de custo penalizada e variância para cada cenário. O valor da função objetivo de PCDRC-MVAR para este exemplo, considerando $\beta = 0,05$ é:

$$MV(K) = 23,33 + 0,05 \cdot (11,09 + 44,09 + 11,09)/3 = 24,44$$

3.5 Solucionando o PCDRC-DevMax e PCDRC-MVar.

O primeiro passo para resolver o problema PCDRC-DevMax consiste em encontrar as soluções ótimas (ou próximas do ótimo) do PCDR, $\delta^s = (\delta_1^s,...,\delta_m^s)$, para cada cenário s. Entretanto, como já foi mostrado no Capítulo 2, métodos de solução exatos para o PCDR só estão disponíveis para problemas de pequeno porte. Portanto, neste trabalho, uma das versões do método *scatter search* desenvolvida no Capítulo 1, é utilizada para obter a melhor solução conhecida de δ^s, para todo $s \in S$. Supondo que os valores de δ^s estão disponíveis, então a resolução do PCDRC-DevMax pode começar. Neste ponto, é interessante analisar como o método de resolução do PCDR determinístico, abordado no Capítulo 1, pode ser adaptado para os modelos PCDRC-DevMax e PCDRC-MVar. Note que a variável de decisão dos três problemas é $K = (a_1, ..., a_m)$. Para um dado valor de K, os três problemas envolvem a resolução do PPPRL, que é feita através da heurística ASP, descrita no Apêndice 1. No caso do PCDR, como existe apenas um cenário, basta resolver o PPPRL uma única vez. Para os problemas com cenários, um PPPRL é resolvido para cada cenário s, o que implica num custo computacional muito maior do que no caso determinístico. Da mesma forma que na heurística de melhoria para o PCDR, o procedimento de eliminação de folgas (PEF) pode ser aplicado aos modelos com cenários: suponha que a disponibilidade do recurso k, seja a_k, e que a quantidade de recurso do tipo k utilizada ao longo do projeto pelo cenário s seja b_{ks}. Se $b_{ks} < a_k$. para todo cenário s, então, a disponibilidade do recurso k é reduzida a $\max_{s=1,...,S}\{b_{ks}\}$.

Uma vez que os três problemas compartilham as mesmas variáveis de decisão, muitas características do método *scatter search* para o PCDR podem ser aproveitadas nos problemas com cenários, como as estratégias de geração de soluções diversas, tipos de subconjuntos de soluções, e métodos de combinação de soluções. A grande diferença entre o PCDR, PCDRC-DevMax e PCDRC-MVar, reside no tipo de função objetivo, e portanto, o método de melhoria deve ser adaptado para a função objetivo de cada modelo.

3.5.1 Heurística de melhoria para o PCDRC-DevMax

Esta seção descreve uma heurística de melhoria que é utilizada no método *scatter search* e na heurística de múltiplos inícios para resolver o PCDRC-DevMax. Esta heurística é baseada em movimentos realizados nas variáveis de disponibilidade de recursos. Um esboço da heurística pode ser visto na Figura 3.2. Seja $(a_1,...,a_m)$ a solução corrente, q o cenário com maior desvio de custo e penalidade relativo a esta solução, e $\delta^q = \left(\delta_1^q,...,\delta_m^q\right)$ a solução ótima (ou melhor solução conhecida) para o cenário q. O método de melhoria consiste em reduzir a distância da solução corrente à solução ótima do cenário q. Para cada recurso do tipo k, $k = 1,...,m$ são gerados movimentos candidatos aumentando a_k de uma unidade se a_k é menor do que δ_k^q, ou reduzindo a_k de uma unidade se a_k é maior do que δ_k^q (Passo 2). No Passo 3, calcula-se o valor da função objetivo da nova solução candidata, e se esta solução apresentar um valor de função objetivo melhor do que a melhor candidata já examinada, x^{best} e *BestSol* são atualizados. Finalmente, após examinar todos os tipos de recursos, a solução corrente é atualizada, recebendo o valor da melhor solução candidata. O método pára se não houver mais melhorias. Um exemplo da aplicação da heurística é apresentado abaixo.

Passo 1. Faça *BestSol* = valor da função objetivo da solução corrente. Faça x^{best} =(a_1,...,a_m).

Enquanto houver melhoria **faça**

{

Para cada tipo de recurso k = 1,..,m

{

Passo 2. Faça (x_1,...,x_m) = (a_1,...,a_m). **Se** $x_k < \delta_k^q$, $x_k = x_k + 1$, **caso contrário**, $x_k = x_k - 1$.

Passo 3. Para a solução candidata (x_1,...,x_m), aplique ASP e PEF. **Faça** *NewObjValue* = valor da função objetivo da solução candidata. **Se** *NewObjValue* < *BestSol*, **faça** x^{best} = (x_1,...,x_m) e *BestSol* = *NewObjValue.*

}

Passo 4. Atualize a solução corrente: (a_1,...,a_m) = x^{best}. Atualize p = cenário com maior valor de desvio.

}

Figura 3.2 Heurística de melhoria para o PCDR-DevMax

A Tabela 3.5 ilustra o efeito do método de melhoria aplicado à solução K=(3,3,4) do exemplo na seção anterior. Para esta solução, o cenário que apresenta o maior desvio é o cenário 2, com valor de desvio máximo igual a 12. A alocação de recursos ótima para o cenário 2 é (4,2,3), dada pela Tabela 3.2. Neste exemplo, a quantidade disponível de recurso do tipo 1 na solução corrrente (a_1 = 3) é menor do que a quantidade recursos disponíveis na solução ótima (δ_1^2 = 4), portanto, segundo a heurística de melhoria, o candidato a ser explorado é: (3+1, 3, 4) = (4,3,4). Para o segundo tipo de recurso temos que $a_k > \delta_2^2$ (3>2), e portanto, a solução candidata é: (3, 3-1, 4). Para k = 3, a solução candidata é (3,3,4-1). Para cada candidato, aplicam-se os procedimentos ASP e PEF. A solução candidata que gera o menor valor de desvio máximo é a solução (4,3,4), com valor de desvio máximo igual a 6. Portanto, esta passa a ser a solução

corrente. O método de melhoria pára quando não houver mais movimentos que impliquem em melhoria da solução corrente.

Tabela 3.5 – Método de melhoria aplicado à solução K = (3,3,4)

Iteração	Desvio máx.	Disponibilidade de recursos	*Makespan* para cenário	Desvio para cenários	Cenário com máx. desvio
1		Solução corrente:			
	12	(3,3,4)	(4,12,4)	(4,12,4)	2
		Movimentos candidatos:			
	6	(4,3,4)	(18,18,15)	(6,4,6)	1 e 3
	12	(3,2,4)	(18,19,19)	(2,10,12)	3
	10	(3,3,3)	(18,19,17)	(2,10,2)	2
		Movimento selecionado:			
	6	(4,3,4)	(18,18,15)	(6,4,6)	1 e 3

3.5.2 Heurística de melhoria para o PCDRC-MVar

Esta seção descreve a heurística de melhoria, para a minimização da função média-variância, utilizada no método *scatter search*. A heurística de melhoria consiste em aumentar ou reduzir, um recurso por vez, de uma unidade, e como já foi mencionado anteriormente, a heurística faz uso dos procedimentos ASP e PEF. Um esboço do procedimento de melhoria pode ser visto na Figura 3.3. Para cada tipo de recurso, o Passo 2 reduz a quantidade disponível de um dos recursos da solução corrente, de uma unidade. Aplica-se ASP e PEF, para obter o valor da função objetivo desta solução candidata. Se a solução candidata tiver menor valor de função objetivo que as demais soluções candidatas já examinadas, então esta passa a ser a melhor solução candidata. O Passo 3 verifica se todos os cenários relativos à solução corrente terminam antes da data de entrega do projeto. Se todos os cenários estão factíveis, então não faz sentido aumentar a disponibilidade de recursos no Passo 3, uma vez que a função objetivo é de minimização, e portanto, a disponibilidade de recursos é reduzida de uma unidade, novamente. Caso contrário, se existir algum cenário infactível, isto é, que resulte num *makespan* maior do que a data de entrega do projeto, então a disponibilidade de um dos recursos é acrescida em uma unidade. Após examinar todos os tipos de recursos, o Passo 4 atualiza a solução corrente com o melhor candidato, caso haja melhoria. A heurística pára, se não houver um candidato com melhor valor de função objetivo do que a solução corrente.

Passo 1. Faça *BesSol* = valor da solução corrente. Faça $x^{best} = (a_1,...,a_m)$.

Enquanto houver melhoria **faça**

{

Para cada tipo de recurso $k = 1,..,m$

{

Passo 2. Faça $(x_1,...,x_m) = (a_1,...,a_m)$. Faça $x_k = x_k - 1$. Para a solução candidata $(x_1,...,x_m)$, aplique ASP e PEF. Faça *NewObjValue* = valor da função objetivo da solução candidata.

Se *NewObjValue* < *BestSol*,

Guarde $x^{best} = (x_1,...,x_m)$ como a melhor solução já encontrada e faça *BestSol* = *NewObjValue*

Passo 3. Faça $(x_1,...,x_m) = (a_1,...,a_m)$.

Se todos os cenários têm soluções factíveis,

$x_k = x_k - 2$,

caso contrário,

$x_k = x_k + 1$.

Para a solução candidata $(x_1,...,x_m)$, aplique ASP e PEF. Faça *NewObjValue* = valor da função objetivo da solução candidata.

Se *NewObjValue* < *BestSol*,

guarde $x^{best} = (x_1,...,x_m)$ como a melhor solução já encontrada e faça *BestSol* = *NewObjValue*.

}

Passo 4. Atualize a solução corrente: $(a_1,...,a_m) = x^{best}$.

}

Figura 3.3 Heurística de melhoria para o PCDR-MVar

CAPÍTULO 4

RESULTADOS COMPUTACIONAIS PARA O PCDRC

Este capítulo apresenta os resultados dos experimentos computacionais do problema de custo de disponibilidade de recursos com cenários. Comparações com heurísticas de múltiplos inícios mostram o ganho de qualidade gerado pelo *scatter search*. Os experimentos também incluem uma análise da influência de parâmetros como fator de data de entrega, fator de penalidade e fator de variância no PCDRC.

4.1 Experimentos computacionais

Os experimentos computacionais foram realizados num PC Pentium III, 667MHz, 256 Mbyte RAM. Todos os procedimentos foram escritos na linguagem de programação C++.

4.2 Geração de problemas testes

O PCDR e o PPPRL possuem alguns parâmetros em comum, o que torna possível adaptar, de forma relativamente simples, problemas testes do PPPRL já existentes, para o PCDR. O programa Progen (Kolisch *et al*., 1995) é um gerador de problemas testes para o PPPRL, que foi utilizado para gerar os problemas testes da biblioteca PSPLIB, envolvendo 4 tipos de recursos e 30, 60, 90 e 120 atividades. Neste trabalho, testes computacionais são realizados com

problemas testes gerados pelo Progen e adaptados para o PCDRC. Problemas testes com 4 recursos e 30, 60 e 90 atividades são diretamente obtidos da biblioteca PSPLIB. O restante dos problemas testes foram gerados pelo programa Progen. Dois importantes parâmetros do Progen são a complexidade de rede (*network complexity – NC*) e o fator de recursos (*resource factor – RF*). *NC* determina o número médio de sucessores imediatos de uma atividade. O fator de recurso varia entre [0,1] e reflete a densidade dos diferentes tipos de recursos que uma atividade requer para ser processada. Por exemplo, se $RF = 1$, 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. A data de entrega D para o projeto foi calculada utilizando o caminho crítico da rede,

$$D = \theta \cdot \max_{s=1,\ldots,S} EF_s$$

tal que θ é o **fator de data de entrega**, e EF_s é o instante de término mais cedo do projeto para o cenário s. Para cada problema teste foram gerados custos C_k aleatoriamente com distribuição uniforme U[1,10], e 10 cenários para as durações das atividades utilizando uma distribuição uniforme U[1,10].

4.3 Resultados computacionais para o PCDRC-DevMax

O objetivo dos testes realizados é analisar o desempenho relativo do *scatter search* (SS) e da heurística de múltiplos inícios com freqüência (MSF), para o PCDRC-DevMax. Como já mencionado no capítulo anterior, a heurística de melhoria para o PCDRC-DevMax é utilizada no MSF e no *scatter search*. Dentre as diversas versões do *scatter search* estudadas no Capítulo 2, a versão (7,3)/ QD/ Dinâmico merece destaque pela boa qualidade dos resultados, além de apresentar valores de distância máxima, em geral, menores do que as demais versões estudadas. Portanto, esta versão será adotada como base para os testes computacionais deste capítulo. Os parâmetros utilizados para (7,3)/ QD/ Dinâmico são: método de combinação G, $Psize = 30$, $g = 10$, $\lambda = 1{,}9$, $b_1 = 7$, $b_2 = 3$. O critério de parada de SS foi definido da seguinte forma. O programa pára após completar 5 iterações do método ou se o tempo computacional exceder duas horas, o que ocorrer primeiro. Entretanto, existem casos em que após duas horas de computação, não foi possível realizar ao menos uma iteração completa do método. Nestes casos, o programa continua

até que ao menos uma iteração completa tenha sido realizada. Observe ainda que podem ocorrer situações nas quais o valor do desvio máximo é igual a zero ($\Psi(K) = 0$), neste caso, o programa é interrompido. O número de soluções avaliadas pelo *scatter search*, para cada problema teste, é guardado e utilizado como critério de parada para o MSF.

Tabela 4.1 Problemas testes

	n	***m***	***DF***
Conjunto 1	30	4	1,20
Conjunto 2	60	4	1,20
Conjunto 3	90	4	1,20
Conjunto 4	60	6	1,20
Conjunto 5	60	8	1,20
Conjunto 6	60	4	1,00
Conjunto 7	60	4	1,40

A Tabela 4.1 descreve os conjuntos de problemas testes considerados neste capítulo. Os parâmetros utilizados para gerar estes problemas testes foram:

- Fator de recurso (*RF*): (0,25), (0,5), (0,75) e (1,0)
- Complexidade de rede (*NC*): (1,5), (1,8) e (2,1)

Para uma dada linha da Tabela 4.1, gerou-se um problema teste para cada valor de *RF* e *NC*, num total de $4 \cdot 3 = 12$ problemas testes por linha. Portanto, considerando as 7 linhas da tabela foram gerados $7 \cdot 12 = 84$ problemas testes.

O primeiro experimento realizado consistiu em comparar o método *scatter search* com a heurística de múltiplos inícios. Todos os problemas testes da Tabela 4.1 foram resolvidos por SS e MSF utilizando o valor $\rho = 50$ para o fator de penalidade. Para todos os problemas testes, o *scatter search* gerou resultados melhores, ou iguais à heurística de múltiplos inícios. Desta forma, os resultados exibidos nas tabelas a seguir referem-se à distância de MSF em relação aos resultados do *scatter search*. Seja distância = 100(*H*1-*H*2)/(*H*2) tal que *H*1 é o valor da função

objetivo da solução encontrada por MSF e $H2$ é o valor da função objetivo encontrada pelo *scatter search*. O desempenho das heurísticas propostas é avaliado pelas seguintes medidas:

- Distância média
- Distância máxima
- Número de empates: número de problemas testes nos quais $H1 = H2$

Tabela 4.2 Distância de MSF em relação a SS de acordo com o fator de recurso, para o PCDRC-DevMax

n	***RF***				**Dist. média (%)**	**Dist. máx. (%)**	**Núm. empates**
	0,25	**0,50**	**0,75**	**1,00**			
30	0,00	14,78	4,74	8,96	7,12	35,43	5
60	0,00	38,74	34,79	23,61	24,29	70,07	3
90	12,71	52,41	70,75	20,85	39,18	127,27	1
Distância média (%)	4,24	35,31	36,76	17,81	23,53		
Distância máx. (%)	36,23	70,07	127,27	36,01		127,27	
Núm. empates	7	1	0	1			9

Tabela 4.3 Comparação de MSF e SS, em relação ao fator de complexidade de rede, para o PCDRC-DevMax

n	***NC***		
	1,50	**1,80**	**2,10**
30	13,49	5,02	2,85
60	15,65	43,14	14,06
90	52,40	37,80	27,34
Distância média (%)	27,18	28,65	14,75
Distância máx. (%)	52,40	43,14	27,34
Núm. empates	2	3	4

As Tabelas 4.2 e 4.3 mostram a distância média, distância máxima e número de empates do MSF, em relação ao *scatter search*, para problemas testes com 4 recursos, 30, 60 e 90 atividades, e fator de data de entrega igual a 1,2 (Conjuntos 1, 2 e 3 da Tabela 4.1). Na Tabela 4.2, para cada valor de *RF* e *n*, a distância média foi calculada sobre 3 problemas testes. Note que, os menores valores de distância média e distância máxima ocorrem nos extremos de *RF*, ou seja, *RF* = 0,25 e *RF* = 1,0. Em particular, o maior número de empates ocorre em *RF* = 0,25. Em geral, a distância média e distância máxima aumentam à medida que o número de atividades cresce. A Tabela 4.3 mostra o desempenho de MSF em relação ao *scatter search*, classificando os problemas testes por fator de complexidade de rede e número de atividades. Para cada valor de *NC* e *n*, os valores da tabela são calculados sobre 4 problemas testes. Para todos os valores de *n*, *NC* = 2,10 apresentou distâncias médias menores e maior número de empates do que *NC* = 1,5 e *NC* = 1,8. Note também que as distâncias máximas crescem à medida que *NC* decresce.

Tabela 4.4 Comparação de MSF e SS, de acordo com número de recursos, para o PCDRC-DevMax

	m		
	4	**6**	**8**
Distância média (%)	18,66	37,59	58,87
Distância máx. (%)	87,25	130,52	198,09
Núm. empates	3	1	0

Os resultados da Tabela 4.4 mostram o desempenho de MSF em relação ao método *scatter search* para 60 atividades, 4, 6 e 8 recursos e *DF* = 1,2 (Conjuntos 2, 4 e 5, da Tabela 4.1). Para cada valor de *m*, a tabela mostra o valor da distância média, distância máxima, e número de empates sobre 12 problemas testes. É possível observar na tabela, que o valor das distâncias média e máxima cresce à medida que *m* cresce. Note também que, para *m* = 8, o número de empates é igual a zero, ou seja, o MSF apresentou resultados piores do que o *scatter search* para os 12 problemas testes. Isto mostra que o método SS gera soluções de melhor qualidade com o

aumento do número de recursos (portanto, número de variáveis) em relação às soluções geradas por MSF.

O próximo experimento consiste em examinar o efeito do fator de data de entrega nos problemas testes com 60 atividades e 4 recursos (Conjuntos 2, 6 e 7, da Tabela 4.1). Os resultados deste experimento são exibidos nas Tabelas 4.5 e 4.6. Para cada valor de *DF*, a Tabela 4.5 exibe a distância média, distância máxima e número de empates em relação a 12 problemas testes. Note que os valores de distância média e número de empates, apresentados na tabela, sugerem que o desempenho do MSF melhora à medida que *DF* aumenta, ou seja, em problemas mais folgados. Provavelmente, o aumento no número de empates ocorre porque problemas mais folgados são mais fáceis de resolver do que problemas mais apertados. É possível notar que se a data de entrega do projeto for muito grande, todos os cenários terão a mesma solução ótima dada pela disponibilidade mínima de recursos,

$$a_k = \max_{i=1,\ldots,n} r_{ik}, k = 1,\ldots,m$$

É claro que, se todos os cenários têm a mesma solução ótima, esta solução é a que minimiza o desvio máximo, no caso igual a zero. Note que a heurística de melhoria proposta consiste em reduzir a distância entre a solução corrente e a solução ótima do cenário com o maior desvio. Portanto, quando os cenários possuem alocações ótimas muito parecidas, a heurística de melhoria, ao tentar reduzir o desvio de um cenário, vai muito provavelmente reduzir o desvio de todos os cenários. À medida que a data de entrega torna-se mais apertada, a disponibilidade ótima de recursos para cada cenário pode ficar mais dissimilar, tornando o problema mais difícil.

Tabela 4.5 Comparação de MSF e SS, de acordo com o fator de data de entrega, para o PCDRC-DevMax

	DF		
	1,00	**1,20**	**1,40**
Distância média (%)	31,02	24,29	18,66
Distância máx. (%)	74,07	70,07	87,25
Núm. empates	1	3	3

A Tabela 4.6 exibe o valor da função objetivo de SS e MSF de acordo com o valor do fator de data de entrega. Note que o desvio máximo diminui quando os problemas são mais folgados.

Tabela 4.6 Média do valor da função objetivo de SS e MSF de acordo com *DF*, para o PCDRC-DevMax

	DF		
	1,00	**1,20**	**1,40**
Média função objetivo SS	87,70	63,79	53,61
Média função objetivo MSF	104,76	83,95	66,96

Foram realizados também alguns testes para avaliar o impacto do fator de penalidade no desvio máximo. Os problemas testes escolhidos para este experimento contêm 60 atividades, 4 recursos e *DF* = 1,0 (Conjunto 6). Os fatores de penalidade testados foram 10, 30, 50, 70, 90 e 110. Os problemas foram resolvidos pelo método *scatter search* e as Figuras 4.1, 4.2 e 4.3 exibem os resultados destes experimentos.

Seja *K* a solução obtida pelo *scatter search*, para um problema teste, então, os seguintes valores são utilizados na Figura 4.1:

- Custo do projeto = $\sum_{k=1,\ldots,m} C_k(a_k)$
- Atraso médio do projeto = $\frac{1}{10}\sum_{s=1,\ldots,10} \max\left(0, \Phi(d_s, K) - D\right)$

A Figura 4.1 mostra as médias do desvio máximo, custo do projeto e atraso médio do projeto, de acordo com o fator de penalidade, para os problemas testes. Por exemplo, se $\rho = 5$, a média do valor da função objetivo (desvio máximo) é igual a 29,92, com um custo médio do projeto igual a 392,77 e atraso médio do projeto igual a 8,94. É possível observar que à medida que o fator de penalidade aumenta, o valor médio do desvio máximo e do custo do projeto apresentam uma tendência de crescimento. A Figura 4.1 mostra também que o atraso médio, em relação ao fator de penalidade, em geral decresce à medida que o fator de penalidade aumenta. Observe que a partir de um determinado valor de penalidade, por exemplo, $\rho > 50$, o custo médio e a média do desvio máximo não variam muito, e o atraso médio do projeto fica próximo de zero. Este tipo de

análise permite ao decisor fazer um estudo sobre o equilíbrio entre atrasar o projeto (e neste caso, arcar com uma penalidade pelo atraso) ou entregar o projeto em dia.

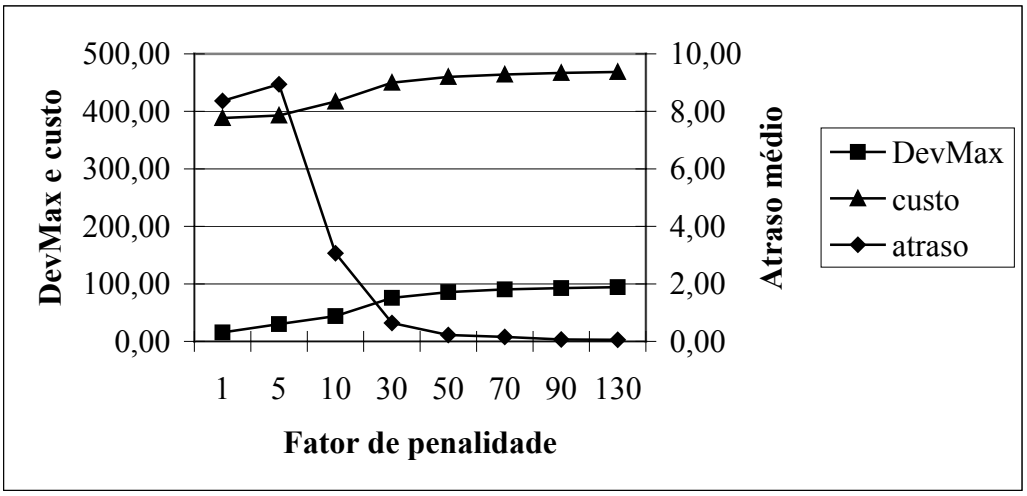

Figura 4.1 Comportamento de SS, em função do fator de penalidade, para o PCDRC-DevMax

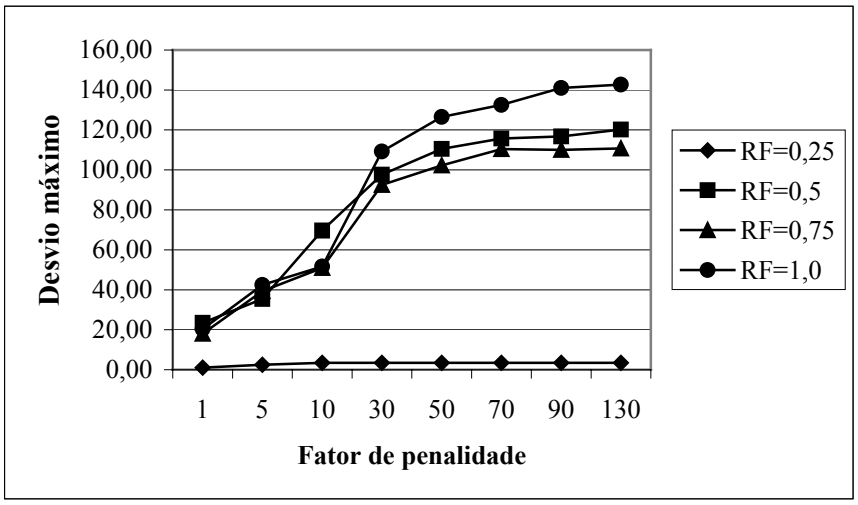

Figura 4.2 Comportamento de SS, em função do fator de penalidade e fator de recurso, para o PCDRC-DevMax

A Figura 4.2 agrupa os problemas testes por *RF* e mostra a média dos desvios máximos de acordo com o fator de penalidade. Note que, a média dos desvios máximos é muito próxima de zero para *RF* = 0,25. Em geral, quanto maior o valor de *RF*, maior é o valor do desvio máximo. Provavelmente, este comportamento se deve ao aumento da dificuldade do problema, uma vez que, nos problemas em que os valores de *RF* são mais altos, as atividades requerem mais tipos de recursos para serem executadas, e conseqüentemente, o conflito por recursos entre os cenários aumenta, tornando mais difícil encontrar uma alocação de recursos que seja boa para todos os cenários. Tome por exemplo, um caso extremo, em que existem *n* atividadades e *m* = *n* recursos, e as atividades não requerem recursos em comum, isto é, a atividade 1 só precisa de r_{11} unidades de recurso do tipo 1 ($r_{1k} = 0$ para $k \neq 1$), a atividade 2 só precisa de r_{22} unidades de recurso do tipo 2 ($r_{2k} = 0$ para $k \neq 2$) , e assim sucessivamente. A solução ótima para este PCDRC-DevMax é $a_i = r_{ii}, i = 1,...,n$, com valor de desvio máximo igual a zero.

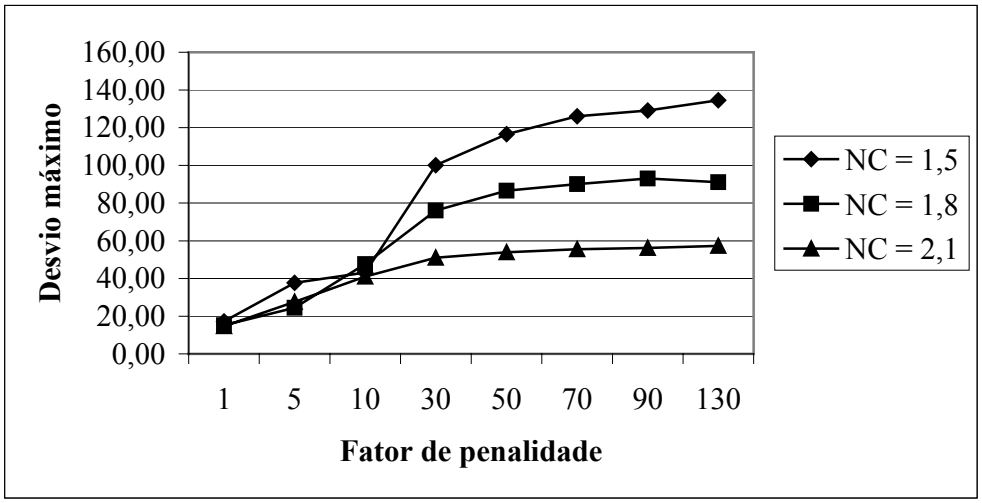

Figura 4.3 Comportamento de SS com relação à complexidade de rede, para o PCDRC-DevMax

A Figura 4.3 ilustra o valor médio do desvio máximo em função do fator de penalidade, de acordo com os valores de complexidade de rede. Note que, em geral, o desvio máximo aumenta à medida que o fator de penalidade e o valor de *NC* aumentam.

Tabela 4.7 Tempo computacional para o SS e MSF, em segundos

	n	***m***	***DF***	**Tempo (segundos)**	**Sol. Aval.**	**Excederam tempo**
Conjunto 1	**30**	4	1,2	3386,16	206838	1
Conjunto 2	**60**	4	1,2	5451,86	109010	9
Conjunto 3	**90**	4	1,2	10874,06	108471	12
Conjunto 4	**60**	6	1,2	6663,80	123388	11
Conjunto 5	**60**	8	1,2	12150,90	210619	12
Conjunto 6	**60**	4	1,0	7229,43	148774	12
Conjunto 7	**60**	4	1,4	7252,70	149133	9

A Tabela 4.7 mostra a média dos tempos computacionais, em segundos, a média do número de soluções avaliadas, e o número de problemas testes que excederam o limite de tempo de duas horas, para cada conjunto de problemas. Note que todos os problemas testes dos Conjuntos 3, 5 e 6 ultrapassaram o limite de 2 horas, isto é, para realizar ao menos uma iteração do *scatter search*, foram necessárias mais de duas horas de computação.

4.4 Resultados computacionais para o PCDRC-MVar

O primeiro experimento realizado para o PCDRC-MVar consiste em comparar o desempenho do *scatter search* com a heurística de múltiplos inícios com freqüência. Como já mencionado no capítulo anterior, a heurística de melhoria para o PCDRC-MVar é utilizada no MSF e no *scatter search*. A versão do *scatter search* (7,3)/ QD/ Dinâmico, que também foi aplicada ao PCDRC-DevMax, é adotada como base para os testes computacionais desta seção. O critério de parada deste experimento, e dos demais realizados nesta seção, é baseado no número de soluções avaliadas pelo *scatter search* para o PCDRC-DevMax. Este experimento utiliza os problemas testes da Tabela 4.1, com fator de penalidade $\rho = 10$, e fator de variância $\beta = 0.05$. Ao comparar o método *scatter search* com a heurística de múltiplos inícios, para todos os

problemas testes da Tabela 4.1, o *scatter search* gerou resultados melhores, ou iguais à heurística de múltiplos inícios. Desta forma, como na seção anterior, os resultados exibidos nas tabelas a seguir referem-se à distância de MSF em relação aos resultados gerados pelo *scatter search.*

Tabela 4.8 Comparação de MSF e SS de acordo com *RF*
para o PCDRC-MVar

n	***RF***				**Dist. média (%)**	**Dist. max. (%)**	**Núm. empates**
	0,25	**0,50**	**0,75**	**1,00**			
30	0,00	1,10	1,40	1,79	1,07	3,79	5
60	0,00	2,58	1,15	1,86	1,40	4,39	4
90	0,87	1,39	1,40	1,18	1,21	3,07	2
Distância média (%)	0,29	1,69	1,32	1,61	1,23	3,75	
Distância max. (%)	1,72	4,39	3,79	2,99			
Núm. empates	7	2	2	0			11

As Tabelas 4.8 e 4.9 mostram o desempenho do MSF em relação ao *scatter search,* para problemas testes com 4 recursos e 30, 60 e 90 atividades (Conjuntos 1, 2 e 3, da Tabela 4.1), classificando os problemas testes por fator de recurso e complexidade de rede, respectivamente. Para cada valor de *RF* e *n*, a Tabela 4.8 mostra a distância média considerando 3 problemas testes. Note que a diferença de desempenho do MSF em relação ao *scatter search* exibidos nesta tabela são bem menores do que os valores de distância do PCDRC-DevMax. Em particular, para os problemas testes de 30 e 60 atividades, com *RF* = 0,25, o MSF empatou com o *scatter search* em todos os problemas testes. Observe que o número de empates decresce à medida que *RF* e *n* aumentam. Em relação à complexidade de rede, é possível notar, na Tabela 4.9, que em média, os maiores valores de distância média e distância máxima ocorreram em *NC* = 1,8. Como já foi observado no PCDRC-DevMax, o maior número de empates ocorreu em *NC* = 2,1.

Tabela 4.9 Comparação de MSF e SS em relação ao fator de complexidade de rede para o PCDRC-MVar

n	***NC***		
	1,50	**1,80**	**2,10**
30	1,38	1,33	0,52
60	0,66	2,10	1,44
90	0,79	1,73	1,11
Distância média (%)	0,94	1,72	1,02
Distância máx. (%)	2,99	4,39	3,03
Núm. empates	3	3	5

A Tabela 4.10 mostra o desempenho de MSF em relação ao SS, de acordo com o número de recursos, para $n = 60$ e $DF = 1{,}2$ (Conjuntos 2, 4 e 5, da Tabela 4.1). Para cada valor de m, os valores da Tabela 4.10 mostram a distância média em relação a 12 problemas testes. Note que a distância média e a distância máxima aumentam de acordo com o número de recursos. O número de vitórias também decresce à medida que o número de recursos aumenta, indicando que a dificuldade dos problemas parece aumentar quando o número de recursos é maior, como já foi observado também no PCDRC-DevMax.

Tabela 4.10 Comparação de MSF e SS em relação ao número de recursos para o PCDRC-MVar

	m			**Dist.**
	4	**6**	**8**	**média (%)**
Distância média (%)	1,40	1,97	3,00	2,12
Distância máx. (%)	4,39	4,57	7,10	
Núm. empates	4	2	1	

Tabela 4.11 Comparação de MSF e SS em relação ao fator de data de entrega para o PCDRC-MVar

	DF			**Distância**
	1,00	**1,20**	**1,40**	**média (%)**
Distância média (%)	1,65	1,40	1,08	1,38
Distância máx. (%)	7,94	4,39	3,43	
Núm. empates	1	4	3	

A Tabela 4.11 mostra o desempenho de MSF em relação ao SS, de acordo com o fator de data de entrega, para problemas com 60 atividades e 4 recursos (Conjuntos de problemas 2, 6 e 7, da Tabela 4.1). Para cada valor de *DF*, a tabela mostra a distância média considerando 12 problemas testes. É possível notar que a distância média e máxima diminuem para problemas mais folgados, como foi observado também no PCDRC-DevMax. Os valores médios da função objetivo dos problemas testes da Tabela 4.11 podem ser vistos na Tabela 4.12. A Tabela 4.12 mostra o valor da função objetivo do *scatter search* e do MSF, de acordo com o valor de *DF*. Note que o valor da função objetivo se reduz à medida que *DF* aumenta, indicando que problemas mais folgados resultam em projetos com custos mais baixos.

Tabela 4.12 Média da função objetivo gerada por SS e MSF para o PCDRC-MVar

	DF		
	1,00	**1,20**	**1,40**
Média função objetivo MSF	466,72	389,53	343,28
Média função objetivo SS	458,97	383,41	339,53

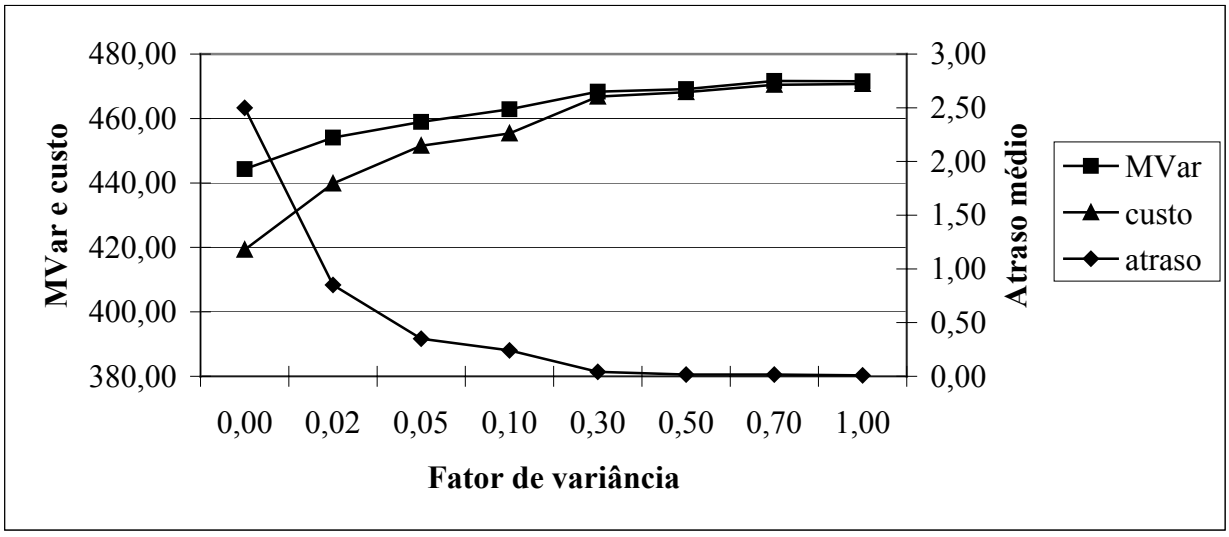

Figura 4.4 Comportamento do SS, em termos do fator de variância, para o PCDRC-MVar

O próximo experimento consiste em mostrar o efeito do fator de variância no PCDRC-MVar. Para isto, foram utilizados problemas testes com 60 atividades, 4 recursos e *DF* = 1,0 (Conjunto 6, da Tabela 4.1). Cada problema teste foi resolvido utilizando o *scatter search*, com os seguintes fatores de variância β: 0, 0,02, 0,05, 0,1, 0,3, 0,5, 0,7 e 1,0, e fator de penalidade, $\rho = 10$. Os resultados deste experimento são mostrados nas Figuras 4.4, 4.5 e 4.6.

A Figura 4.4 mostra os valores médios, da função objetivo, custo do projeto e atraso, considerando 12 problemas testes, para cada fator de variância. Note que quando o fator de variância é igual a zero, o *scatter search* minimiza o custo de disponibilidade médio. Observando o gráfico, é possível notar que à medida que o fator de variância aumenta, o valor da função objetivo e o custo do projeto aumentam, ao contrário do atraso médio, que diminui. Isto é, quanto maior a aversão do decisor ao risco, mais alto é o custo do projeto, por exemplo, se o valor do fator de variância for $\beta = 0{,}05$, o projeto pode ser realizado com um custo de 450, gerando um atraso médio, por cenário, de 0,4. Por outro lado, se o decisor tem grande aversão a risco, então escolherá um valor mais alto para o fator de variância, como $\beta = 0{,}5$, que resultará numa solução com pouca variância (sem atraso), mas precisará pagar um custo mais alto pelo projeto, no valor de 470. É interessante observar que, quando o fator de variância é maior do que 0,3, o atraso médio do projeto é praticamente zero e o valor da média-variância e custo do projeto

são muito parecidos (por volta de 470). Este tipo de análise pode auxiliar o decisor a encontrar um equilíbrio adequado entre minimizar o risco e minimizar o custo do projeto.

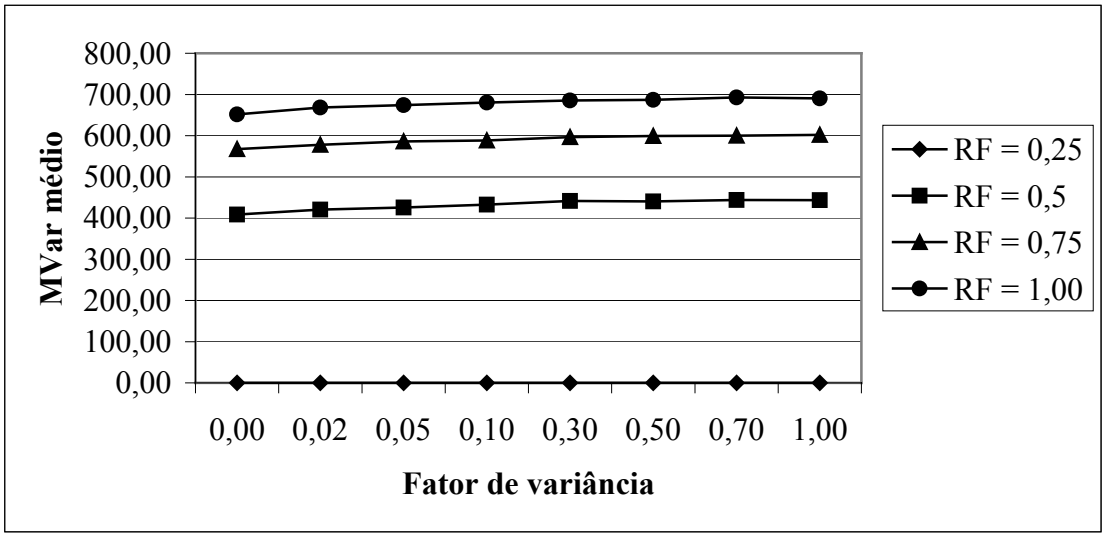

Figura 4.5 Comportamento de SS de acordo com fator de variância e fator de recurso para o PCDRC-MVar

A Figura 4.5 mostra a média dos valores da função objetivo do PCDRC-MVar, de acordo com o fator de variância e fator de recurso. Note que, o valor da função objetivo, em geral, cresce com o fator de variância, e para um dado fator de variância, problemas com fatores de recursos mais altos, resultam em valores de função objetivo mais altos. Para problemas com RF = 0,25, o valor da função objetivo é praticamente constante, porque nestes problemas, as soluções possuem um valor de atraso e variância muito baixos, mesmo quando β é pequeno.

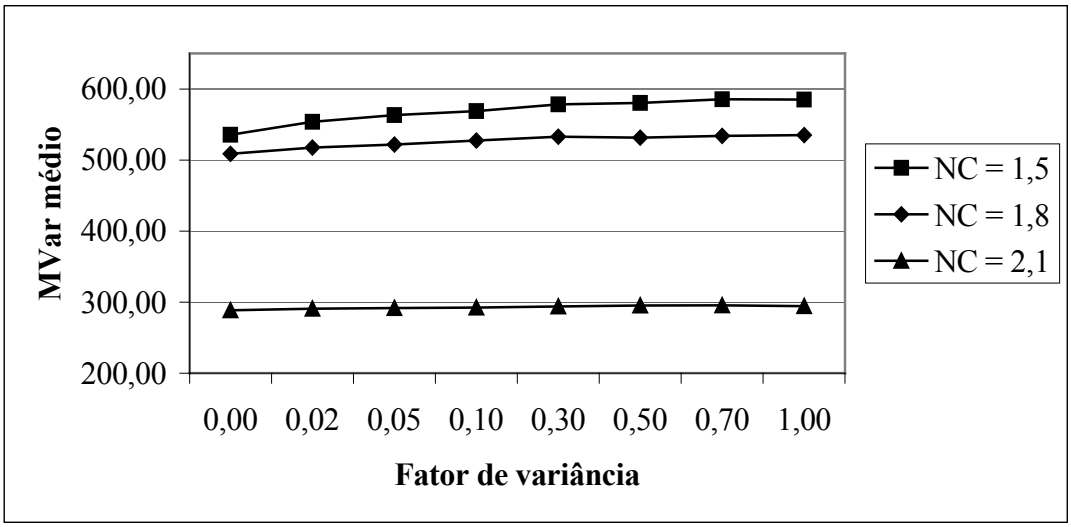

Figura 4.6 Comportamento do SS de acordo com fator de complexidade de rede, para o PCDRC-MVar

A Figura 4.6 mostra o valor da função objetivo do PCDRC-MVar de acordo com o fator de complexidade de rede e fator de variância. Note que problemas com maior valor de *NC* geram um valor de média-variância menor. Isto também ocorre para o problema PCDRC-DevMax (veja Figura 4.3).

O próximo experimento consiste em estudar as diferenças entre o PCDRC-DevMax e PCDRC-MVar, solucionados pelo *scatter search*. Nas Tabelas 4.13 – 4.16, foram considerados problemas testes com 60 atividades, 4 recursos e *DF* = 1,0 (Conjunto 6, da Tabela 4.1), com $\rho = 10$. O primeiro experimento consistiu em comparar o valor do desvio máximo gerado pelo PCDRC-MVar, ou seja, quando a média-variância é minimizada, com o desvio máximo gerado pelo PCDRC-DevMax. Considere a solução gerada pelo PCDRC-DevMax. Os valores do desvio máximo ($\Psi(K)$) gerados pelo *scatter search* são exibidos na segunda coluna da Tabela 4.13. Suponha agora que os mesmos problemas testes sejam resolvidos com a função objetivo do PCDRC-MVar, com os fatores de variância exibidos na segunda linha da Tabela 4.13. Para cada valor de fator de variância, calcula-se o desvio máximo da solução obtida ao se resolver o PCDRC-MVar. Com exceção dos problemas em que *RF* = 0,25, os resultados são bem distintos. É interessante notar que para *RF* = 0,25 e fator de variância 0,5 e 0,7, as soluções do PCDRC-

MVar apresentam, em média, um desvio máximo menor do que o PCDRC-DevMax. Para valores de *RF* maiores, principalmente *RF* = 1,0, a solução do PCDRC-MVar apresenta um valor de desvio máximo que, em geral, cresce com o fator de variância.

Tabela 4.13 Desvio máximo para PCDRC-DevMax e PCDRC-MVar.

RF	**DevMax desvio**	**Fator de variância - β**							
		0,00	**0,02**	**0,05**	**0,10**	**0,30**	**0,50**	**0,70**	**1,00**
0,25	3,45	3,67	3,67	3,95	3,95	3,45	3,16	3,16	3,45
0,50	69,64	79,81	85,46	91,61	98,16	115,44	117,04	117,95	119,93
0,75	50,97	61,52	73,83	94,26	97,28	111,35	111,59	116,01	114,95
1,00	51,71	69,45	100,39	120,38	126,08	139,49	143,49	147,16	146,97

A Tabela 4.14 mostra o atraso médio da solução gerada pelo PCDRC-DevMax (Coluna 2) e PCDRC-MVar (Colunas 3-10). Note que, para *RF* > 0,5, o atraso médio do PCDRC-DevMax foi maior do que o atraso médio gerado para PCDRC-MVar.

Tabela 4.14 Atraso médio para o PCDRC-DevMax e PCDRC-MVar.

RF	**DevMax atraso**	**Fator de variância - β**							
		0,00	**0,02**	**0,05**	**0,10**	**0,30**	**0,50**	**0,70**	**1,00**
0,25	0,00	0,03	0,03	0,03	0,03	0,00	0,00	0,00	0,00
0,50	2,10	2,23	0,60	0,40	0,30	0,07	0,00	0,03	0,00
0,75	3,87	2,97	1,33	0,47	0,20	0,03	0,07	0,00	0,03
1,00	6,30	4,77	1,43	0,50	0,43	0,07	0,00	0,03	0,00

A Tabela 4.15 mostra o custo médio da solução gerada pelo PCDRC-DevMax (Coluna 2) e PCDRC-MVar (Colunas 3-10), de acordo com o fator de recurso. É interessante notar que o custo médio das soluções do PCDRC-DevMax não difere muito do custo obtido ao minimizar o custo esperado (PCDRC-MVar com $\beta = 0$).

Tabela 4.15 Custo do projeto para o PCDRC-DevMax e PCDRC-MVar

RF	DevMax custo	Fator de variância - β							
		0,00	**0,02**	**0,05**	**0,10**	**0,30**	**0,50**	**0,70**	**1,00**
0,25	150,27	149,04	149,04	149,32	149,32	150,27	149,98	149,98	150,27
0,50	392,89	386,21	408,98	415,12	421,67	438,96	440,55	441,46	443,44
0,75	531,45	537,74	557,64	578,08	581,09	595,16	595,40	599,82	598,76
1,00	594,21	604,13	643,76	663,75	669,45	682,86	686,86	690,53	690,34

A Tabela 4.16 mostra o valor da variância, em média, obtida pelo PCDRC-DevMax (Coluna 2) da tabela e a variância obtida pelo PCDRC-MVar. O valor de variância expresso na Tabela 4.16 foi calculado como:

- $variância = \frac{1}{10}\sum_{s=1,\ldots,10}(MD(K) - CP(K,s))^2$

tal que $CP(K,s)$ é o valor da função de custo do PCDR penalizada, para o cenário s e $MD(K)$ refere-se à média dos valores de $CP(K,s)$. Para maiores detalhes, veja Capítulo 3, Seção 3.1. Exceto pelos problemas com $RF = 0{,}25$, o valor da variância apresentado por PCDRC-DevMax é muito maior do que os valores de variância gerados pelo PCDRC-MVar. Para os 12 problemas testes deste experimento, isso mostra que, de fato, as duas funções objetivo podem dar resultados bastante distintos. É interessante notar que, para $RF = 0{,}25$ e $RF = 0{,}5$, minimizar a média apenas ($\beta = 0$) resultou em uma variância maior do que a minimização do desvio máximo.

Tabela 4.16 Variância para o PCDRC-DevMax e PCDRC-MVar

RF	PCDRC DevMax Variância	Fator de variância - β							
		0,00	**0,02**	**0,05**	**0,10**	**0,30**	**0,50**	**0,70**	**1,00**
0,25	0,00	3,00	3,00	3,00	3,00	0,00	0,00	0,00	0,00
0,50	1455,00	1838,33	282,00	128,00	80,33	6,00	0,00	3,00	0,00
0,75	1106,67	1076,33	345,33	68,67	53,33	3,00	6,00	0,00	3,00
1,00	2449,00	2213,67	512,33	112,33	67,00	6,00	0,00	3,00	0,00

Os próximos resultados procuram mostrar detalhadamente, a solução gerada pelo *scatter search* para um dos problemas testes estudados. A Tabela 4.17 mostra a solução do PCDRC-DevMax para um problema teste do Conjunto 6, com fator de penalidade igual a 10. A Tabela

4.18 mostra a solução para o mesmo problema teste, mas resolvendo o modelo PCDRC-MVar, com β igual a 0,01. Para cada cenário, as Tabelas 4.17 e 4.18 mostram o valor do custo do projeto somado ao valor de penalização ($CP(K,s)$), custo ótimo de cada cenário, desvio e a diferença entre o custo médio e o custo penalizado ($(MD(K)-CP(K,s))^2$). Os dados exibidos na Tabela 4.17 são ilustrados na Figura 4.7. As barras na figura indicam o valor do custo ótimo de cada cenário, e as setas verticais mostram o valor do desvio de cada cenário. Note que os cenários 9 e 6 mostram situações extremas, em que o menor custo e o maior custo, respectivamente, são atingidos. A influência destes dois cenários pode ser notada na Tabela 4.17, que mostra que os dois maiores valores de variância ocorrem no Cenário 6 e no Cenário 9. Note que na Tabela 4.17 os desvios são mais uniformes, enquanto na Tabela 4.18, a variância é mais uniforme, devido às características das funções objetivo. Na Tabela 4.18, é possível observar também que o Cenário 6 apresenta um desvio negativo, isto é, a solução gerada pelo *scatter search* para o PCDRC-MVar, ultrapassa a data de entrega do problema, pois a penalização neste cenário é relativamente baixa, em relação aos demais. Este exemplo mostra que o decisor deve ficar atento à presença de cenários extremos, que podem dificultar a resolução do problema.

Tabela 4.17 Exemplo de aplicação do PCDRC-DevMax num problema teste

Cenários	**Custo penalizado**	**Custo ótimo de cada cenário**	**Desvio**	$(MD(K)-CP(K,s))^2$
1	731,29	692,52	38,77	900,00
2	681,29	641,22	40,07	400,00
3	681,29	652,59	28,70	400,00
4	731,29	712,71	18,58	900,00
5	681,29	643,80	37,49	400,00
6	791,29	762,00	29,29	8100,00
7	651,29	627,30	23,99	2500,00
8	731,29	685,59	45,70	900,00
9	631,29	583,50	47,79	4900,00
10	701,29	657,90	43,39	0,00

Tabela 4.18 Exemplo de aplicação do PCDRC-MVar num problema teste

Cenários	Custo penalizado	Custo ótimo de cada cenário	Desvio	$(MD(K) - CP(K,s))^2$
1	718,87	692,52	26,35	361,00
2	678,87	641,22	37,65	441,00
3	688,87	652,59	36,28	121,00
4	728,87	712,71	16,16	841,00
5	688,87	643,80	45,07	121,00
6	748,87	762,00	-13,13	2401,00
7	678,87	627,30	51,57	441,00
8	688,87	685,59	3,28	121,00
9	678,87	583,50	95,37	441,00
10	698,87	657,90	40,97	1,00

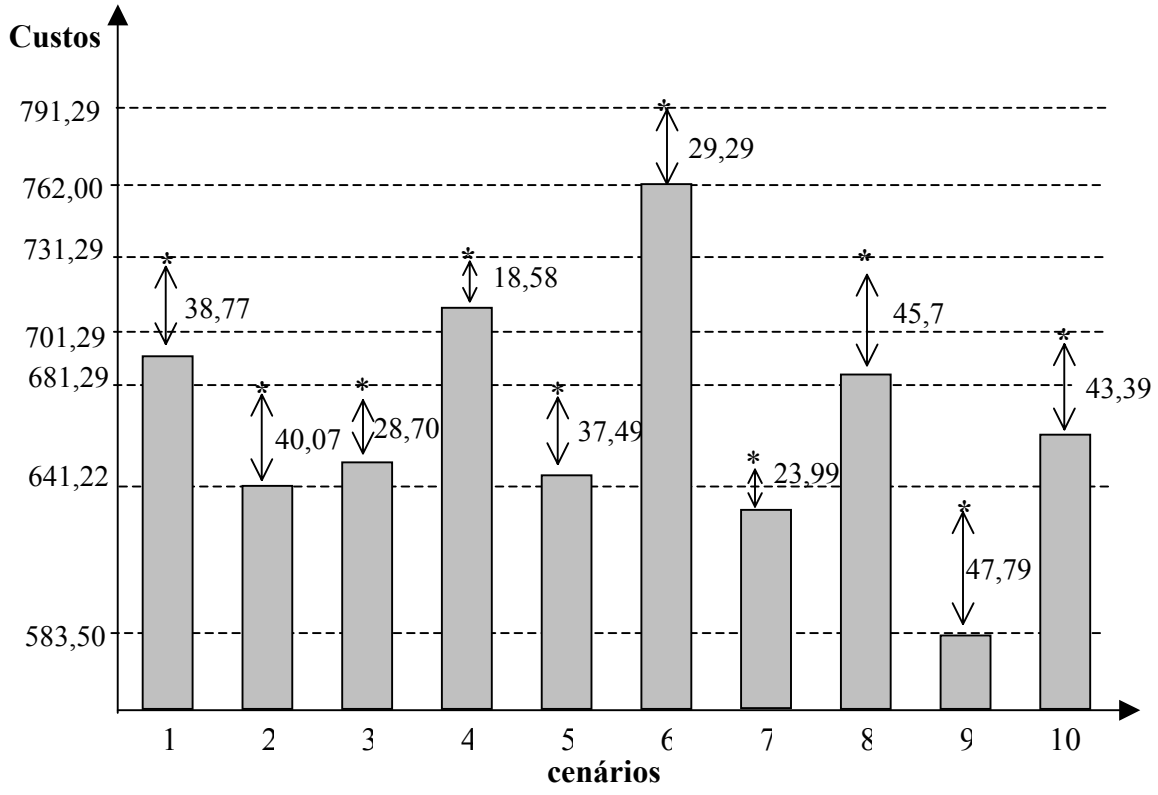

Figura 4.7 Exemplo de um problema teste do PCDRC-DevMax

4.5 Conclusões

Os Capítulos 3 e 4 deste trabalho, abordaram o problema de programação de projetos com incerteza nas durações das atividades. Enquanto muitos estudos envolvendo incerteza otimizam um resultado médio, o objetivo deste trabalho foi utilizar uma abordagem para controlar a variabilidade da solução. Para isto, foi proposto um procedimento de solução baseado em otimização robusta, no qual as durações das atividades são representadas como um conjunto de possíveis cenários e procura-se uma solução que seja robusta em relação aos cenários. Dois modelos para o PCDRC foram estudados, o modelo de minimização do desvio máximo e o modelo de minimização da média-variância. O método *scatter search,* proposto para o PCDR, foi adaptado para resolver os dois modelos. Foi implementada também, uma heurística simples de múltiplos inícios, com a finalidade de mostrar o ganho de qualidade gerado por uma heurística mais complexa, como o *scatter search*. O *scatter search* encontrou soluções melhores ou iguais à heurística de múltiplos inícios, em todos os problemas testes resolvidos, mostrando ser um método adequado para resolver o PCDRC-MVar e PCDRC-DevMax.

Os resultados computacionais também mostraram que, apesar das estratégias de minimização do desvio máximo e minimização de média-variância serem alternativas para lidar com risco, estas estratégias podem resultar em soluções bastante distintas, no sentido em que, a minimização do desvio máximo não resulta, necessariamente, em uma solução com baixa variância, e a minimização da média-variância não resulta, necessariamente, em uma solução com baixo desvio máximo. Portanto, a escolha adequada de uma função objetivo deve refletir a atitude do decisor com relação ao risco.

CAPÍTULO 5

CONCLUSÕES

Neste trabalho foi proposto um método de resolução, baseado na heurística *scatter search*, para o problema de programação de projetos conhecido como Problema de Custo de Disponibilidade de Recursos. Três modelos do problema foram abordados: PCDR, PCDRC-DevMax e PCDRC-MVar. Um resumo dos principais resultados do trabalho são discutidos a seguir.

O método *scatter search* proposto mostrou-se capaz de resolver PCDRs de grande porte, e gerar soluções de boa qualidade. Em problemas de pequeno porte, o *scatter search* foi comparado a soluções exatas e limitantes inferiores, e encontrou soluções ótimas ou próximas do ótimo para um número significativo de problemas testes. O método também obteve melhores resultados do que as heurísticas de múltiplos inícios em 92,5% dos problemas testes de médio porte. Aplicações do *scatter search*, na literatura, são relativamente recentes e em geral, apresentam implementações mais básicas do método, sem utilizar estratégias avançadas. Neste trabalho, estratégias mais avançadas foram exploradas, e comparadas com implementações mais básicas. O método *scatter search* mostrou-se robusto, uma vez que seu desempenho não é muito sensível com relação às versões testadas.

Dois modelos de otimização robusta, PCDRC-DevMax e PCDRC-MVar, foram propostos. O método *scatter search,* proposto para o PCDR, foi adaptado para resolver os dois modelos. O *scatter search* encontrou soluções melhores ou iguais à heurística de múltiplos

inícios, em todos os problemas testes resolvidos, tanto para o PCDRC-MVar como para o PCDRC-DevMax.

Os resultados computacionais também mostraram que, apesar das estratégias de minimização do desvio máximo e minimização de média-variância serem alternativas para lidar com risco, estas estratégias podem resultar em soluções bastante distintas, no sentido em que, a minimização do desvio máximo não resulta, necessariamente, em uma solução com baixa variância, e a minimização da média-variância não resulta, necessariamente, em uma solução com baixo desvio máximo. Portanto, a escolha adequada de uma função objetivo deve refletir a atitude do decisor com relação ao risco.

O algoritmo do *scatter search* programado neste trabalho não foi planejado para ser eficiente em termos de velocidade computacional, portanto, seria interessante, no futuro, desenvolver um algoritmo mais otimizado, principalmente para os modelos com cenários, que demandam um esforço computacional maior. Em relação ao *scatter search* para o PCDR, existem novas estratégias promissoras que poderiam ser aplicadas tais como, partição das soluções em *clusters* definidos por uma distância, e *path relinking*. Outro tema a ser explorado, é o desenvolvimento de um algoritmo genético para comparação com o método *scatter search*. Pesquisas futuras envolvendo incerteza incluem o estudo de funções objetivo como a minimização da soma ponderada do custo médio com o desvio máximo, e o desenvolvimento de modelos de otimização robusta para outros problemas da área de programação de projetos.

REFERÊNCIAS

Bell, C.E., Han, J., 1991. A new heuristic solution method in resource-constrained project scheduling, Naval Research Logistics 38 315-331.

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: classification and complexity, Discrete Applied Mathematics 5 (1) 11-24.

Boctor, F., 1990. Some efficient multi-heuristic procedures for resource constrained project scheduling, European Journal of Operational Research 49 3-13.

Boctor, F., 1996. Resource constrained project scheduling by simulated annealing, International Journal of Production Research, 34 (8) 2335-2351

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

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research 107 272-288.

Campos, V., Glover, F., Laguna, M., Martí, M., 2001. An experimental evaluation of a scatter search for the linear ordering problem, Journal of Global Optimization 21 (4) 397-414.

Campos, V., Laguna, M., M. Martí M, 2003. Context-independent scatter and tabu search for permutation problems, submetido para publicação.

Cavalcante, C.C.B., 1998. O problema de escalonamento com restrições de mão-de-obra. Dissertação (Ciência da Computação) - Universidade Estadual de Campinas

Cavalcante, C.C.B., Souza, C.C., Savelsbergh, M.W.P., Wang, Y., Wolsey, L.A., 2001. Scheduling projects with labor constraints, Discrete Applied Mathematics 112 27-52.

Cavique, L., Rego, C., Themido, I., 2001. A scatter search algorithm for the maximum clique problem. In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, pp: 227-244.

Cho, J.-H., Kim, Y.-D., 1997. A simulated annealing algorithm for resource constrained project scheduling problems, Journal of the Operational Research Society 48 736-744.

Christophides, N., Alvarez-Valdes, R., Tamarit, J.M., 1987. Project scheduling with resource-constraints: classification and complexity, Discrete Applied Mathematics 5 11-24.

Cooper, D., 1976. Heuristics for scheduling resource-constrained projects: An experimental investigation, Management Science 22 (11) 1186-1194.

Daniels, R.L., Kouvelis, P., 1995. Robust scheduling to hedge against processing time uncertainty in single-stage production, Management Science 41 (2) 363-376.

Davis, E., Patterson, J., 1975. A comparison of heuristic and optimum solutions in resource-constrained project scheduling, Management Science 21 944-955.

Demeulemeester, E., 1995. Minimizing resource availability costs in time-limited project networks, Management Science 41 (10) 1590-1598.

Demeulemeester, E., Herroelen, S., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Management Science 38 (12) 1803-1818.

Demeulemeester, E., Herroelen, W., 1997. New benchmark results for the resource constrained project scheduling problem, Management Science 43 (11) 1485-1492.

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

Escudero, L.F., 1993. Production planning via scenario modelling, Annals of Operations Research 43 311-335.

Escudero, L.F., Quintana, F.J., Salmerón, J., 1999. CORO, a modeling and an algorithmic framework for oil supply, transformation and distribution optimization under uncertainty, European Journal of Operational Research 114 638-656.

Gagnon, M., D'Avignon, G., Boctor, F., 2002. Minimisation du coût de disponibilité des ressources d'un projet sous contrainte de temps par une adaptation de la méthode tabou, INFOR 40 (3) 277-304.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints, Decision Sciences 8 156-166.

Glover, F., 1998. A template for scatter search and path relinking. In: Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (Eds.), Artificial Evolution, Lecture Notes in Computer Science 1363, Springer, pp. 13-54.

Glover , F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking, Control and Cybernetics 39 (3) 653-684.

Glover , F., Laguna, M., Martí, R., 2002. Scatter search and path relinking: Foundations and advanced designs, to appear in New Optimization Techniques in Engineering, G. Onwubolu (Eds) Springer-Verlag.

Golenko-Ginzburg D., Gonik A., 1997. Stochastic network project scheduling with non-consumable limited resources, International Journal of Production Economics 48 29-37.

Golenko-Ginzburg D., Gonik A., 1998. A heuristic for network project scheduling with random activity durations depending on the resource allocation, International Journal of Production Economics 55 149-162.

Greistorfer, P., 2003. A tabu scatter search metaheuristic for the arc routing problem, Computers & Industrial Engineering 44 (2) 249-266.

Gutierrez, J.G., Kouvelis, P., 1995. A robustness approach to international sourcing, Annals of Operational Research 59 165-193.

Hamiez, J-P., Hao, J-K., 2002. Scatter search for graph coloring, Lecture Notes in Computer Science 2310 168-179.

Hartmann, S., 1998. A competitive genetic algorithm for resouce-constrained project scheduling, Naval Research Logistics 45 733-750.

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

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, European Journal of Operational Research 127 (2) 394-407.

Herroelen, W., De Reyck B., Demeulemeester, E., 1998. Resource-constrained project scheduling: a survey of recent developments, Computers & Operations Research 25 (4) 279-302.

Jain, A.S., Meeran, S., 2002. A multi-level hybrid framework applied to the general flow-shop scheduling problem, Computers & Operations Research 29 (13) 1873-1901.

Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operations Research 90 320-333.

Kolisch, R., Drexl, A., 1996. Adaptive search for solving hard project scheduling problems, Naval Research Logistics 43 (1) 23-40.

Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems, Management Science 41 (10) 1693-1703.

Kolisch, R., Hartmann, S., 1998. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling: Recent Models, Algorithms, and Applications, Kluwer Academic Publishers, pp. 147-178.

Krajewski, L.J., Ritzman, L.P., 1996. Operations management: Strategy and Analysis, Addison-Wesley Publishing Company.

Laguna, M., 1998. Applying robust optimization to capacity expansion of one location in telecommunications with demand uncertainty, Management Science 44 101-110.

Laguna, M., 2002. Scatter search. In: Pardalos, P.M., Resende, M.G.C., (Eds.), Handbook of Applied Optimization, Oxford University Press, 183-193.

Laguna, M., Armentano, V.A., 2002. Lessons from applying and experimenting with scatter search, to appear in Adaptive memory and evolution: Tabu search and scatter search, César Rego e Bahram Alidaee (Eds).

Laguna, M., Martí, R., 2002. Experimental testing of advanced scatter search designs for global optimization of multimodal functions, submetido para publicação.

Laguna, M., Martí, R., 2003**.** Scatter Search: Methodology and Implementations in C, Kluwer Academic Publishers, Boston.

Lee, J.-K., Kim, Y.-D., 1996. Search heuristics for resource constrained project scheduling, Journal of the Operational Research Society 47 678-689.

Li, R.-Y., Willis, J., 1992. An iterative scheduling technique for resource-constrained project scheduling, European Journal of Operational Research 56 370-379.

Malcolm, S. A., Anandalingam, G., 2000. Robust optimization for power systems capacity expansion planning in large countries, Relatório Técnico, University of Maryland.

Malcolm, S.A., Zenios, A.S., 1994. Robust optimization for power systems capacity expansion under uncertainty, Journal of the Operational Research Society 45 1040-1049.

Martí, R., Laguna, M., Campos, V., 2002. Scatter search vs. genetic algorithms: an experimental evaluation with permutation problems. To appear in: Rego, C., Alidaee, B., (Eds.), Adaptive memory and evolution: Tabu search and scatter search, Kluwer Academic Publishers.

Markowitz, H., 1959. Portfolio selection, efficiency diversification of investments, Yale University Press, New Haven, Conn.

Martí, R., Lourenço, H., Laguna, M., 2000. Assigning proctors to exams with scatter search. In: Laguna, M., Velaverde, J.L.G., (Eds.), Computing Tools for Modeling, Optimization and Simulation, Kluwer Academic Publishers, pp. 215-227.

Mingozzi, A., Maniezzo, V., Ricciardelli S., Bianco, L., 1998. An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation, Management Science 44 5 714-729.

Moder, J.J., Phillips, C.R., Davis, E.W., 1983. Project management with CPM, PERT, and precedence diagramming, Van Nostrand Reinhold, New York.

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

Möhring, R.H., Stork, F., 1998. Linear preselective policies for stochastic project scheduling, Technical report no. 612/1998, Technische Universitä Berlin, Department of Mathematics, Germany.

Mulvey, J.M., Vanderbei R.J., Zenios, S.A., 1995. Robust optimization of large-scale systems, Operations Research 43 (2) 264-281.

Patterson, J.H., 1984. A comparison of exact approaches for solving the multiple constrained resource project scheduling problem, Management Science 30 854-867.

Radermacher, F. J., 1985. Scheduling of project networks, Annals of Operations Research 4 227-252.

Rangaswamy, B., 1998. Multiple resource planning and allocation in resource-constrained project networks, Ph.D. thesis, Graduate School of Business, University of Colorado.

Rego, C., Leão, P., 2001. A scatter search tutorial for graph-based permutation problems, submetido para publicação.

Sampson, S.E., E.N. Weiss , E.N., 1993. Local search techniques for the generalized resource constrained project scheduling problem, Naval Research Logistics 40 665-675.

Thomas, P., Salhi, S., 1997. An investigation into the relationship of heuristic performance with network resource-characteristics, Journal of the Operational Research Society 48 1 34-43.

Tsai, Y.-W., Gemmill D.D., 1998. Using tabu search to schedule activities of stochastic resource-constrained projects, European Journal of Operational Research 111 129-141.

Ulusoy, G., Özdamar, L., 1989. Heuristic performance and network/resource characteristics in resource-constrained project scheduling, Journal of the Operational Research Society 40 12 1145-1152.

Valls, V., Laguna, M., Lino P., Pérez A., Quintanilla S., 1998. Project scheduling with stochastic activity interruptions. In: Weglarz, J. (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer Academic Publishers, pp. 333-353.

Yu, C-S., Li, H-L., 2000. A robust optimization model for stochastic logistic problems, International Journal of Production Economics 64 385-397.

APÊNDICE A

O PROBLEMA DE PROGRAMAÇÃO DE PROJETOS COM RECURSOS LIMITADOS

A1. Descrição do problema

Este apêndice apresenta o problema de programação de projetos com recursos limitados (PPPRL) e descreve a heurística *ASP* utilizada no método de melhoria. Considere um projeto com n atividades sujeito a relações de precedência $(i, j) \in H$, em que as atividades 1 e n são atividades artificiais que indicam o início e fim do projeto, respectivamente. Cada atividade i tem duração d_i e requer r_{ik} unidades do recurso do tipo k ($k = 1, ..., m$) durante o seu processamento. Seja A_t, o conjunto das atividades em progresso durante o intervalo de tempo $(t\text{-}1,t]$, D, a data de entrega do projeto, e a_k, a disponibilidade do recurso do tipo k. As variáveis do problema são os instantes de término f_i das atividades. Um modelo conceitual para o PPPRL é dado a seguir.

Minimizar f_n

sujeito a

$$f_j \geq f_i + d_j \qquad \forall (i, j) \in H$$

$$f_1 = 0$$

$$\sum_{A_t} r_{ik} \leq a_k \qquad k = 1,...,m \quad \text{e} \quad t = 1,....,f_n$$

Como pode ser observado no modelo acima, a função objetivo consiste em minimizar o *makespan.* A primeira restrição refere-se às relações de precedência entre as atividades, a segunda restrição indica que o projeto deve começar no instante zero, e a última restrição do modelo limita a quantidade de recurso do tipo k que pode ser utilizada pelas atividades no intervalo (t-1,t].

O PPPRL é *NP-hard* (Blazewicz *et al.*, 1983) e tem sido estudado extensivamente na literatura. Revisões bibliográficas recentes sobre métodos exatos e heurísticos podem ser encontradas em Herroelen *et al*. (1998), Brucker *et al*. (1999) e Kolisch e Hartmann (1998). Para resolver o modelo de otimização robusta, o algoritmo *scatter search* envolve a resolução de diversos PPPRLs ao longo de sua execução, sendo necessário então utilizar um método de resolução de baixo custo computacional para estes subproblemas. Dentre os diversos métodos propostos na literatura para o PPPRL, as heurísticas do tipo *single-pass* ou *multi-pass*, como a heurística *ASP* (*adaptive search procedure)* proposta por Kolisch e Drexl (1996), são uma boa opção, pois apresentam um bom compromisso entre soluções de qualidade e tempo computacional. Estas heurísticas são descritas neste apêndice.

A2. Problemas Testes

Dois conjuntos de problemas testes, bastante conhecidos na literatura de programação de projetos, são o conjunto proposto por Patterson (1984) e o conjunto *KSD* proposto por Kolisch *et al.* (1995).

O conjunto de Patterson consiste de 110 problemas testes, cujas soluções ótimas são conhecidas, e contém problemas com até 3 tipos de recursos e número de atividades variando de 7 a 50. Este conjunto de problemas tem sido utilizado para avaliar o desempenho de muitos algoritmos exatos e heurísticas para o PPPRL.

Kolisch *et al*. (1995) propõem um novo gerador de problemas testes para o PPPRL, mais complexo do que os 110 problemas de Patterson, chamado de *ProGen*, que permite gerar problemas que satisfazem certos parâmetros. Este parâmetros são utilizados para determinar características do problema, como por exemplo, o número médio de sucessores ou predecessores das atividades. Isso permitiu a criação do conjunto de problemas *KSD*, que consiste de 480

problemas testes (30 atividades, 4 tipos de recursos). Posteriormente, este conjunto de problemas testes foi estendido para problemas com 60, 90 e 120 atividades, originando a biblioteca PSPLIB.

A3. Heurísticas para o PPPRL

Tabela 1. - Desempenho de algumas heurísticas em relação aos 110 problemas de Patterson.

Autor	Tipo de heurística	Desvio médio da sol. ótima em (%)	Tempo (s)
Kolisch e Drexl (1996) – ASP	1 realização	3,71	0,005a
	10 realizações	1,82	0,04a
	100 realizações	1,22	0,25a
	500 realizações	0,71	1,14a
Bell e Han (1991)	Melhoria	2,6	28,4b
Sampson e Weiss (1993)	Melhoria	1,98	10,2c
Hartmann (1998)	Algoritmo genético	0,00	5,0d
		110 na otimalidade	5,0d
Lee e Kim (1996)	Busca tabu	0,75	17,0c
	Simulated annealing	0,57	17,0c
Cho e Kim (1997)	Simulated annealing	0,15	18,4c
		103 na otimalidade	
Boctor (1996)	Simulated annealing	0,015	-
		109 na otimalidade	

a Tempo médio de CPU num IBM PC 486.

b Tempo médio de CPU num Macintosh plus.

c Tempo médio de CPU num Pentium 60 MHz.

d Tempo máximo de CPU num Pentium 133 MHz.

O número de heurísticas propostas para o PPPRL é bastante vasto, e encontram-se desde regras de prioridade, até metaheurísticas como busca tabu, *simulated annealing* e algoritmos

genéticos. A Tabela 1 apresenta o desempenho de algumas heurísticas propostas na literatura para o PPPRL quando aplicadas ao conjunto de problemas de Patterson.

É possível observar na Tabela 1, que os melhores resultados computacionais são obtidos por metaheurísticas, como o algoritmo genético de Hartmann (1998) e o *simulated annealing* de Boctor (1996). Entretanto, é importante lembrar que neste trabalho, o PPPRL é apenas um subproblema do problema de custo de disponibilidade de recursos (PCDR), e será resolvido diversas vezes durante a execução do algoritmo *scatter search*. Por este motivo, estamos interessados em heurísticas mais rápidas do que metaheurísticas, apesar dos melhores resultados serem obtidos por estas. Neste contexto, os procedimentos baseados em regras de prioridade parecem ser uma escolha adequada. As heurísticas descritas na Tabela 1 são desenvolvidas a partir de métodos para geração de programas (*schedule generation scheme*), discutidos a seguir.

Esquemas de Geração de Programas

Os dois esquemas de geração de programas descritos a seguir constroem um programa factível (isto é, as relações de precedência e restrições de recursos são respeitadas), estendendo, passo a passo, um programa parcial. Os esquemas podem ser diferenciados pela maneira como incrementam o programa parcial. Enquanto o esquema serial utiliza avanço (ou incremento) por atividades, o esquema paralelo utiliza avanço no tempo.

Esquema Serial. O esquema serial para geração de programas consiste de $l = 1,2,...,n$ estágios, e em cada estágio, uma atividade é selecionada e programada. Associados a cada estágio existem dois conjuntos disjuntos de atividades, S_l e D_l. S_l contém as atividades que já foram programadas e pertencem ao programa parcial, e D_l contém as atividades que ainda não foram programadas e cujos predecessores pertencem a S_l, isto é, D_l contém as atividades que estão disponíveis para programação com respeito às relações de precedência. Em cada estágio, uma atividade do conjunto D_l é selecionada e programada o mais cedo possível, obedecendo suas restrições de recursos e precedência. Essa atividade é então retirada do conjunto D_l e inserida em S_l. Novas atividades são colocadas no conjunto de decisão D_l. O algoritmo termina no estágio $l = n$, quando todas as atividades estão em S_n.

Esquema Paralelo. O esquema paralelo consiste de no máximo n estágios nos quais um conjunto de atividades, que pode ser vazio, é programado. Cada estágio l está associado a um instante t_l, tal que $t_q \leq t_l$ para $q < l$. O instante t_l corresponde ao instante de término de uma atividade e, conseqüentemente, ao instante em que unidades de recurso são liberadas. As atividades podem ser divididas então em dois conjuntos, em relação a t_l : o conjunto C_l das atividades que já foram programadas e completadas até o instante t_l, e o conjunto IP_l, que consiste das atividades que foram programadas, mas que ainda não terminaram de ser processadas. Finalmente, temos o conjunto de decisão D_l que contém todas as atividades que ainda não foram programadas e que estão disponíveis para programação com respeito às restrições de precedência e recursos. Em cada estágio l, seleciona-se sucessivamente uma atividade pertencente a D_l que será programada no instante t_l, e atualiza-se o conjunto D_l. As atividades são inseridas em IP_l. à medida que vão sendo programadas. Se não existirem mais atividades que possam começar no instante t_l, respeitando as restrições de factibilidade, então o esquema paralelo passa para o próximo estágio. O programa parcial de cada estágio é formado pelas atividades de C_l e IP_l. O algoritmo pára após todas as atividades terem sido programadas.

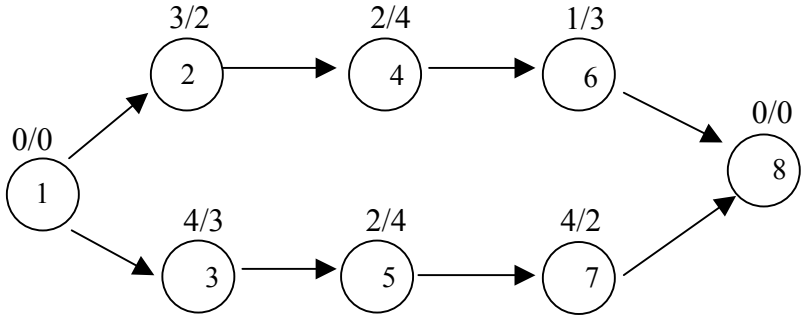

Figura 1 Exemplo de um projeto

Exemplos do esquema de geração serial e paralelo são apresentados nas Tabelas 2 e 3, respectivamente, para o projeto da Figura 1. A Figura 1 ilustra um projeto com n = 8 atividades, k = 1 tipo de recurso com disponibilidade de a_1 = 4 unidades. Os números acima de cada atividade

correspondem à duração e quantidade de recursos utilizados pela atividade i (d_i / r_{ik}). Na Tabela 2, a primeira linha corresponde às iterações do esquema de geração de programa, a segunda linha ilustra o conjunto de decisão D_l para cada iteração l, e a terceira linha da tabela indica a atividade escolhida para ser programada na iteração l. Tanto no esquema serial quanto no esquema paralelo, a seleção da atividade do conjunto D_l é feita de forma aleatória. Coincidentemente, os esquemas serial e paralelo geraram um programa igual, apresentado no diagrama de Gantt da Figura 2.

No exemplo do esquema serial, inicialmente, programa-se a atividade artificial 1, e portanto, o conjunto D_2 é formado pelas atividades 2 e 3. Seleciona-se, aleatoriamente, a atividade 3 para ser programada. Esta atividade deve ser programada o mais cedo possível obedecendo as restrições de recurso, e portanto, a atividade 3 é programada no instante 0. No estágio 3, o conjunto D_3 é formado pelas atividades 2 e 5, e a atividade 5 é selecionada para ser programada. Como a atividade 5 precisa de quatro unidades de recurso para ser executada, então ela é programada no instante 4 e o algoritmo prossegue para o estágio 4. No estágio 4, o conjunto D_4 contém as atividades 2 e 7. A atividade 2, que utiliza 2 unidades de recurso, é selecionada para ser programada e começa a ser processada no instante 6, terminando no instante 9. No estágio seguinte, D_5 contém as atividades 4 e 7, e a atividade selecionada para ser programada é a atividade 7 que deve ser alocada o mais cedo possível, respeitando as restrições de recurso, ou seja, a atividade 7 pode ser alocada no instante 6. No estágio 6, a única atividade que é factível em termos de relação de precedência, é a atividade 4, e portanto, esta atividade é programada no instante 10. O algoritmo prossegue, programando as atividades 6 e 8, nos estágios 7 e 8, respectivamente.

Tabela 2 Exemplo do esquema de geração serial

l	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{D_l}$	{1}	{2,3}	{2,5}	{2,7}	{4,7}	{4}	{6}	{8}
i	1	3	5	2	7	4	6	8

Na Tabela 3, a primeira linha corresponde às iterações do esquema de geração de programa, a segunda linha refere-se ao instante de tempo t_l, a terceira linha mostra o conjunto de

decisão D_l para cada iteração l, e a quarta linha da tabela indica a atividade escolhida para ser programada na iteração l.

No exemplo do esquema de geração paralelo, inicialmente, o conjunto D_2 é formado pelas atividades 2 e 3. A atividade 3 é selecionada para ser programada no instante $t_2 = 0$. Como não existem mais recursos suficientes para programar a atividade 2 no instante $t_2 = 0$, o algoritmo prossegue para o estágio 3. No estágio 3 o instante de tempo é atualizado para $t_3 = 4$ (instante de tempo mais cedo no qual recursos são liberados), e D_3 é composto pelas atividades 2 e 5. A atividade 5 é então selecionada para ser programada no instante 4 e como não existem recursos disponíveis para programar a atividade 2, prossegue-se para o estágio 4. No estágio 4 o instante de tempo é atualizado para $t_4 = 6$ e $D_4 = \{2, 7\}$. A atividade 2 é então selecionada para ser programada em $t_4 = 6$ e $D_4 = \{7\}$. Como ainda existem recursos disponíveis para processar a atividade 7, então a atividade 7 também é programada no instante 6, e o algoritmo prossegue para o próximo estágio. No estágio 5, o instante de tempo é atualizado para $t_5 = 9$, próximo instante em que uma atividade termina de ser processada e recursos são liberados. Neste estágio, o conjunto D_5 é vazio, pois não existe nenhuma atividade factível em termos de precedência e recursos. O algoritmo prossegue para o estágio 6, e o instante de tempo é atualizado para $t_6 = 10$ (próximo instante de tempo em que uma atividade termina de ser processada) e $D_6 = \{4\}$. A atividade 4 é então alocada no instante 10 e finalmente, no estágio 7, a atividade 6 é programada no instante 12.

Tabela 3. - Exemplo do esquema de geração paralelo

l	**1**	**2**	**3**	**4**	**4**	**5**	**6**	**7**	**8**
t_l	0	0	4	6	6	9	10	12	15
D_l	{1}	{2,3}	{2,5}	{2,7}	{7}	{ }	{4}	{6}	{8}
i	1	3	5	2, 7	7	-	4	6	8

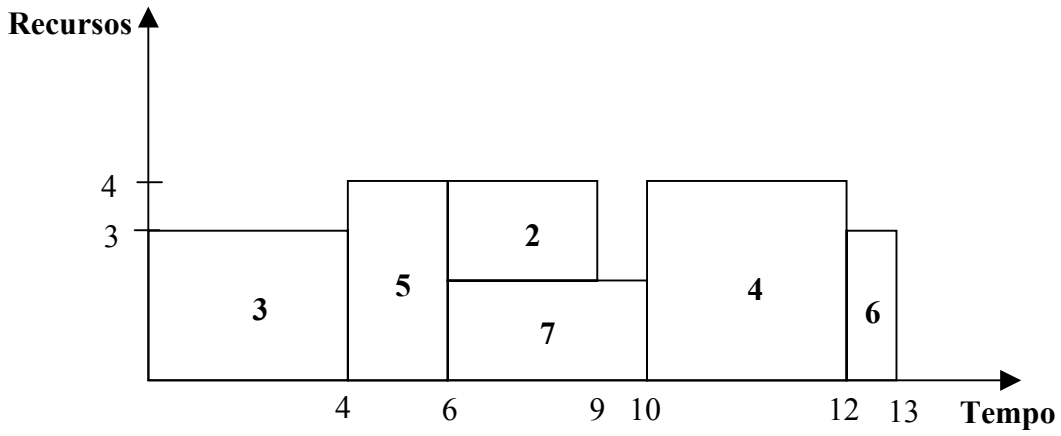

Figura 2 Diagrama de Gantt para o programa gerado pelos esquemas serial e paralelo

Heurísticas construtivas

Nesta seção são apresentados métodos heurísticos baseados em regras de prioridade para o PPPRL. Tais heurísticas são bastante populares por serem, em geral, fáceis de implementar, intuitivas e rápidas. Basicamente, estes métodos são uma combinação de regras de prioridade com um esquema de geração de programas (paralelo ou serial), e podem ser classificados *multi-pass* ou *single-pass*.

Métodos *single-pass*

Métodos *single-pass* utilizam o esquema paralelo ou serial, combinado com uma única regra de prioridade, para gerar somente um programa. Diversos trabalhos contendo revisões bibliográficas sobre regras de prioridade podem ser encontrados na literatura, como Davis e Patterson (1975), Boctor (1990) e Kolisch (1996). Nestes trabalhos, os autores concluem que não existe uma regra de prioridade que apresente um desempenho superior às demais em todos os problemas testes. As regras de prioridade mais conhecidas para o PPPRL são citadas na Tabela 4.

Tabela 4. Desempenho das regras de despacho

Regra		**O**	***v(j)***
MSLK	*(minimum job slack)*	min	$LF_j - EF_j$
LFT	*(latest finish time)*	min	LF_j
RAN	*Seleciona as atividades aleatoriamente*		
GRPW	*(greatest rank positional weight)*	max	$d_j + \sum_{i \in S_j} d_i$
LST	*(latest start time)*	min	$LF_j - d_j$
MTS	*(most total successors)*	max	$\left\|\overline{S}_j\right\|$
SPT	*(shortest processing time)*	min	d_j
WCS	*(worst case slack)*	min	$LF_j - d_j - \max_{(i,j)\in AP}\{E(i,j)\}$

LF_j = instante de término mais tarde

EF_j = instante de término mais cedo.

S_j = conjunto das atividades que já foram programadas

$AP = \{(i, j) \in D_l \times D_l \mid i \neq j\}$: conjunto de todos os pares de atividade que estão no conjunto de decisão D_l

$E(i, j)$ = instante de início mais cedo da atividade *j*, factível em termos de precedência e recursos se *i* começa a ser processada no instante t_l.

$\left|\overline{S}_j\right|$ = número de sucessores imediatos da atividade *j*.

Na tabela acima encontram-se algumas das regras mais conhecidas para o PPPRL. A regra MSLK atribui maior prioridade às atividades com menor folga, que é dada pela diferença

entre o instante de término mais tarde e o instante de término mais cedo. O instante de término mais cedo de uma atividade corresponde ao instante de tempo mais cedo no qual uma atividade pode terminar, respeitando apenas as restrições de precedência, sem levar em conta as restrições de recursos. O instante de término mais tarde corresponde ao instante de tempo mais tarde no qual uma atividade pode terminar, respeitando as restrições de precedência e um limitante superior para o *makespan*. Outra regra citada na Tabela 4 que utiliza o conceito de folga para calcular os valores de prioridade é a regra WCS. Neste caso, a folga de uma atividade corresponde à diferença entre o instante de início mais tarde LF_j e o pior instante de início caso a atividade j seja processada após uma atividade pertencente a D_l. Algumas regras levam em consideração o número de sucessores imediatos das atividades na hora da programação, como ocorre com a regra MTS, que procura dar prioridade maior para atividades que têm um maior número de sucessores.

Tabela 5. Desempenho das regras de despacho

Ranking de regras de prioridade (Kolisch , 1996)
$LST \succ LFT \succ\succ MTS \succ MSLK \succ\succ GRPW$
"$\succ$" denota melhor e "$\succ\succ$" denota significativamente melhor

A Tabela 5 apresenta o resultado obtido por Kolisch (1996) ao comparar o desempenho de diversas regras de prioridade da literatura. Neste trabalho foram gerados 360 problemas testes com 30 atividades e 4 recursos utilizando o programa *ProGen*. Em outro artigo (Kolisch e Drexl, 1996), os autores comparam também a regra WCS com as regras citadas na Tabela 5, e concluem que esta regra tem um desempenho melhor do que as demais quando o método paralelo é utilizado. As regras WCS e LFT são utilizadas na heurística ASP, descrita mais adiante.

Métodos multi-pass

Devido à dificuldade em encontrar uma regra de prioridade que apresente um bom desempenho para todos os problemas, muitos trabalhos baseados em métodos *multi-pass* foram

propostos. Isto é, ao invés de utilizar somente uma regra e uma única passagem, métodos do tipo *multi pass* realizam *Z* passagens simples para gerar uma amostra de no máximo *Z* soluções, dentre as quais, a melhor delas é escolhida.

Os métodos *multi-pass* podem ser divididos em duas categorias: método com múltiplas regras de prioridade (*multi priority rule approach*) (Boctor (1990) e Li e Willis (1992)) que emprega um esquema de geração de programas e diversas regras de prioridade e amostragem (*sampling*) que utiliza um esquema de geração de programas e um componente aleatório associado a uma regra de prioridade para gerar diversos programas.

Métodos com múltiplas regras de prioridade. Neste método, diversos programas são gerados utilizando uma regra de prioridade distinta a cada passagem. Boctor (1990), por exemplo, emprega 7 regras diferentes em seu trabalho. Ao invés de utilizar *NumRegras* regras de prioridade para gerar *NumRegras* programas, é possível gerar diversos programas utilizando-se combinações convexas de *I* valores de prioridades $v(j) = \sum_{i=1,\ldots,I} w_i \cdot v_i(j)$ com $w_i \geq 0$ para todo *i* e $\sum_{i=1,\ldots,I} w_i = 1$. Exemplos de tais abordagens podem ser encontrados em Ulusoy e Özdamar (1989) e Thomas e Salhi (1997). Uma outra abordagem envolvendo múltiplas regras de prioridade pode ser encontrada em Li e Willis (1992), que propõem o chamado *método iterativo*. Este método emprega um esquema de programação regressiva (*backward scheduling*), no qual as atividades são alocadas uma a uma, começando da última para a primeira. Nesta heurística, programação progressiva e regressiva são realizadas alternadamente, até que não ocorram mais melhorias no *makespan* do projeto. Os valores de prioridades são obtidos utilizando-se o instante de início ou instante de término do último programa gerado.

Método de amostragem. Neste método, ao invés de calcular um valor de prioridade para cada atividade, calcula-se uma probabilidade $p(j)$, que corresponde à probabilidade da atividade *j*, pertencente ao conjunto de decisão D_l, ser selecionada. Dependendo do modo como as probabilidades são calculadas pode-se classificar o método de amostragem em amostragem aleatória (*random sampling*), amostragem aleatória tendenciosa (*biased random sampling*) e amostragem aleatória tendenciosa baseada em desvio (*regret based biased random sampling*).

Amostragem aleatória: associa a cada atividade no conjunto de decisão a mesma probabilidade $p(j) = 1 \Big/ |D_l|$.

Amostragem aleatória tendenciosa: emprega os valores de prioridade diretamente para obter as probabilidades, $p(j) = v(j) \Big/ \left(\sum_{i \in D_l} v(i) \right)$. Uma aplicação deste tipo de amostragem pode ser encontrada no trabalho de Cooper (1976).

Amostragem aleatória tendenciosa baseada em desvio: utiliza os valores de prioridade $v(j)$ indiretamente, através de valores de desvio (*regret*). O valor de desvio r_j compara o valor de prioridade da atividade corrente *j* com o pior dos valores de prioridade dentre todas as atividades no conjunto de decisão, ou seja,

$$r_j = \begin{cases} v(j) - \min_{i \in D_l} v(i), & \text{se} \quad O = \max \\ \max_{i \in D_l} v(i) - v(j), & \text{se} \quad O = \min \end{cases}$$

tal que $O = \min$ signifca que a regra de prioridade ordena as atividades do conjunto D_l em ordem decrescente e $O = \max$ significa que a regra de prioridade ordena as atividades do conjunto D_l em ordem crescente. A probabilidade p_j da atividade *j* ser selecionada é então dada por:

$$p(j) = \frac{(r_j + 1)^{\alpha}}{\sum_{i \in D_l} (r_i + 1)^{\alpha}} \qquad (1)$$

tal que o parâmetro α controla a tendenciosidade da escolha. Note que $\alpha = 0$ implica numa seleção por distribuição uniforme. Observe na equação acima que a adição da constante "1" ao valor de desvio r_j assegura que a probabilidade de selecionar uma atividade é maior do que zero para todas as atividades.

Kolisch (1996) estuda os métodos *single-pass* e métodos de amostragem utilizando tanto o esquema de geração de programa serial quanto o paralelo. Em seu trabalho, ele mostra que a classificação de desempenho das regras de prioridade não dependem do esquema de geração de programa escolhido, isto é, a classificação do desempenho das regras de prioridade da Tabela 3 é a mesma tanto para o esquema paralelo como para o esquema serial. Segundo o autor, o esquema de geração de programa paralelo origina melhores resultados para métodos do tipo *single-pass*, e *multi-pass* com amostragem pequena, bem como para problemas com fator de recursos alto. O fator de recursos *RF,* varia entre [0,1] e reflete a densidade dos diferentes tipos de recursos utilizados por uma atividade. Por exemplo, se $RF = 1$, cada atividade requer todos os m tipos de recursos enquanto para $RF = 0$, nenhuma atividade requer qualquer tipo de recurso. O esquema serial é superior quando se utiliza uma amostragem maior e quando os problemas testes apresentam um valor baixo para *RF*. Observe que estes resultados são utilizados na heurística proposta por Kolisch e Drexl (1996) descrita a seguir.

Heurística ASP

Dentre as heurísticas *multi-pass* da literatura, a heurística ASP (*Adaptive Search Procedure*) de Kolisch e Drexl (1996) merece destaque pelos bons resultados computacionais gerados. Os autores propõem uma heurística adaptativa para resolver o PPPRL que utiliza o método de *amostragem aleatória tendenciosa baseada em desvio*, descrita acima. A heurística é um híbrido de regra de prioridade e técnicas de busca aleatória. O método utiliza dois tipos de adaptações: seleção de um espaço de solução apropriado e controle da área de busca do espaço de solução escolhido.

1 - Seleção de um espaço de solução apropriado. Trata-se de uma escolha apropriada do esquema de geração de programas utilizado, paralelo ou serial. Através de testes computacionais os autores concluem que a superioridade de um esquema de programação depende de dois fatores: fator de recurso *RF* e número de realizações *Z*. Para problemas com $RF \leq 0,75$ e $Z > 0$ utiliza-se o esquema serial, caso contrário, utiliza-se o esquema paralelo. Além do esquema de geração de programas, é necessário escolher uma regra de prioridade associada a cada esquema. Através de

testes computacionais, os autores selecionaram a regra LFT para o método serial e a regra WCS para o método paralelo (veja Tabela 4).

2 - O segundo tipo de adaptação consiste em determinar o parâmetro de tendência α (veja equação (1)) e portanto, o tamanho do espaço solução pesquisado. Na primeira passagem do algoritmo, Z = 1 e o valor de $\alpha = \infty$ e portanto o espaço de solução é restrito à solução determinística. À medida que Z aumenta, o valor de α deve decrescer, e desta forma, amplia-se o tamanho do espaço de solução percorrida.

Os autores também propõem a aplicação de limitantes inferiores para eliminar soluções ou programas (*scheduling*) parciais de baixa qualidade e desta forma, reduzir o custo computacional. A heurística foi testada nos 110 problemas propostos por Patterson (1984), apresentando um desvio médio em relação à solução ótima de 1,22% e 0,71%, para 100 e 500 realizações (Z = 100 e Z = 500), respectivamente, como mostrado na Tabela 1.

APÊNDICE B

ALGORITMOS EXATOS PARA OS PROBLEMAS PCDR E PPPRL

B1. Algoritmo exato para o PCDR

Este apêndice apresenta uma descrição do algoritmo exato proposto por Demeulemeester (1995) para resolver o problema de custo de disponibilidade de recursos. O algoritmo começa determinando um limitante inferior para a quantidade de recursos disponíveis. Uma vez fixada a disponibilidade de recursos, um PPPRL (problema de programação de projeto com recursos limitados) 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 PCDR. Os PPPRLs são solucionados utilizando-se um *branch-and-bound* proposto por Demeulemeester e Herroelen (1992), descrito na Seção B2 deste apêndice.

No algoritmo exato de Demeulemeester (1995), inicialmente, é proposto um procedimento baseado na resolução de diversos PPPRLs, com o objetivo de gerar pontos eficientes, definidos a seguir.

Definição: O ponto i com disponibilidade de recursos $(a_1,..., a_m)$ é um ponto eficiente se não existe outro ponto j com disponibilidade de recursos $(a'_1,..., a'_m)$ no espaço de solução tal que todo $a'_k \leq a_k$ para $k = 1,..., m$.

Uma vez que estes pontos são definidos, tenta-se resolver o PPPRL que corresponde ao ponto eficiente de menor custo. Se nenhuma solução factível puder ser encontrada com esta disponibilidade de recursos, este ponto eficiente é cortado do espaço de solução. Novos pontos eficientes são definidos, para os quais, aumenta-se a disponibilidade de um dos tipos de recurso em uma unidade, e a busca continua, procurando resolver o PPPRL que corresponde ao ponto eficiente mais barato. Este processo é então repetido até que uma solução factível é encontrada pela primeira vez (o projeto termina dentro da data de entrega e nenhuma disponibilidade de recursos é violada em qualquer instante durante a execução do projeto). A disponibilidade de recursos correspondente ao ponto eficiente que gerou a primeira solução factível encontrada constitui a solução ótima.

Exemplo gráfico

Para esclarecer melhor a estratégia de resolução do PCDR, um exemplo simples descrito em Demeulemeester (1995), com dois tipos de recursos é apresentado. O custo de disponibilidade de recursos é dado por $C(a_1,a_2) = a_1 + 3a_2$, tal que a_1 e a_2 denotam a disponibilidade de recursos do tipo 1 e 2, respectivamente. Inicialmente, o espaço de soluções corresponde ao quadrante que está acima do eixo a_1 e à direita do eixo a_2 (veja Figura 1). O espaço de soluções pode ser reduzido ao se calcular a quantidade de recursos máxima necessária sobre todas as atividades, para qualquer tipo de recurso. Suponha que a necessidade máxima de recurso de qualquer atividade é 2 para o primeiro tipo de recurso e 2 para o segundo tipo de recurso, obtém-se uma fronteira do espaço de soluções delimitada pelas linhas A e B, que correspondem aos limitantes inferiores correntes da disponibilidade dos dois recursos. É possível melhorar estes limitantes resolvendo PPPRLs com um único tipo de recurso. Suponha que não seja possível encontrar um programa factível com um *makespan* menor ou igual à data de entrega do projeto, para $a_1 = 2$ e a_2 = infinito, mas que um programa factível pode ser encontrado quando $a_1 = 3$. Neste caso, a linha C pode ser introduzida, indicando que nenhuma solução factível para o problema de decisão pode ser encontrada com uma disponibilidade de recurso do tipo 1 menor do que 3. Assumindo que existe um programa factível quando $a_2 = 2$, nenhum limitante novo é introduzido. Nesta situação, a noção de ponto eficiente é aplicada. O espaço de soluções que fica do lado direito da linha C e

acima da linha B, pode ser caracterizado pelo ponto $x1$ com coordenadas (3,2) e um custo de disponibilidade de recursos igual a 9. Este é um ponto eficiente que domina (tem menor custo) todos os pontos delimitados pelas linhas B e C.

Nesta fase, o algoritmo *branch-and-bound* para o PPPRL é aplicado no ponto $x1$. Assumindo que nenhuma solução factível pode ser encontrada para $x1$, este ponto é cortado do espaço de solução e dois novos pontos eficientes são definidos: ponto $x2$ com coordenadas (4,2) e um custo de disponibilidade de recursos igual a 10 e ponto $x3$ com coordenadas (3,3) e um custo de 12.

O próximo passo consiste em resolver o PPPRL para o ponto $x2$, que é o de menor custo: se nenhum programa factível existe para este ponto, o ponto $x2$ é cortado do espaço de soluções e um novo ponto eficiente é introduzido, o ponto $x4$ com coordenadas (5,2) e um custo igual a 11.

O próximo ponto eficiente a ser considerado é o ponto $x4$: suponha que não existe um programa factível para este ponto, portanto, ele é cortado do espaço de soluções. Um novo ponto eficiente é introduzido: o ponto $x5$ com coordenadas (6,2) e custo igual a 12. O outro ponto eficiente $x3$ também tem custo 12. Arbitrariamente, escolha o ponto $x3$ primeiro e assumindo que um programa factível possa ser encontrado para este ponto, a busca pode ser concluída com uma solução ótima que tem um custo de disponibilidade de recursos igual a 12.

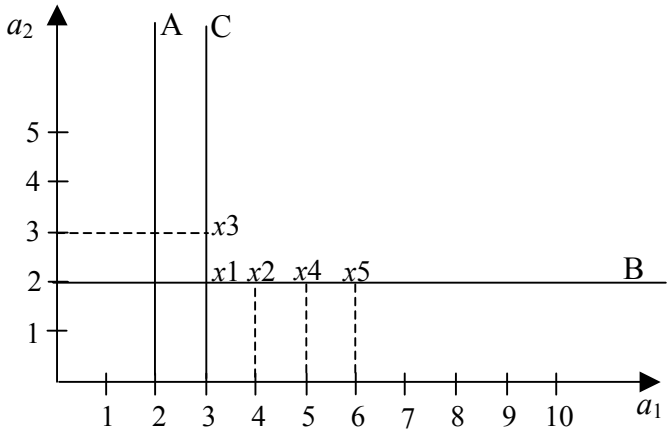

Figura 1. Exemplo da estratégia de solução para o PCDR

B2. *Branch-and-bound* para o PPPRL

Processo de busca

O *branch-and-bound* descrito nesta seção foi proposto por Demeulemeester e Herroelen (1992). O algoritmo realiza uma busca em profundidade, e é uma extensão do método proposto por Christophides *et al*. (1987). Note que, quando este método é utilizado no procedimento do PCDR descrito na Seção B1, não é preciso, necessariamente, chegar ao final da solução do *branch-and-bound*, uma vez que deseja-se saber apenas se a disponibilidade de recursos resulta num programa que é factível com relação à data de entrega ou não. De fato, se em algum ponto do procedimento for encontrado um limitante superior para o *makespan* menor do que a data de entrega do projeto, o *branch-and-bound* pode ser interrompido. Isso significa que esta disponibilidade de recursos para o PCDR é factível. Um outro caso que provoca a interrupção do *branch-and-bound* é descrito a seguir. Se o método encontra um limitante inferior para o *makespan*, e este limitante inferior é maior do que a data de entrega do projeto, isto significa que esta disponibilidade de recursos é uma solução infactível para o PCDR. Neste caso, o *branch-and-bound* é interrompido, e o procedimento de solução do PCDR prossegue explorando outra solução eficiente.

Os nós da árvore de busca do *branch-and-bound* correspondem a programas (*schedules*) parciais em que instantes de término são, temporariamente, designados a um subconjunto de atividades do projeto. Os programas parciais são factíveis, satisfazendo tanto relações de precedência, como restrições de recursos. *Programas parciais* PS_t são considerados somente naqueles instantes de tempo t que correspondem ao término de uma ou mais atividades do projeto. Neste esquema, as atividades começam a ser processadas o mais cedo possível, obedecendo as restrições de recursos e precedência. Um programa parcial PS_t no instante t consiste, portanto, do conjunto de *atividades programadas* temporariamente. Decisões de programação são temporárias no sentido em que atividades programadas temporariamente podem ser atrasadas como resultado de decisões tomadas em estágios posteriores no processo de busca. Atividades não-programadas no instante t são aquelas para as quais as decisões de programação

temporária não foram tomadas ainda. No instante t, o programa parcial PS_t correspondente contém algumas atividades que foram terminadas e outras que ainda estão em progresso. As atividades que foram terminadas têm instantes de término menores ou iguais a t e são colocadas no conjunto F_t de atividades completadas no instante t. As atividades que não foram terminadas ainda, pertencem ao conjunto S_t de atividades em progresso no instante t. O conjunto das atividades incompletas U_t no instante de tempo t contém todas as atividades que não pertencem a F_t.

Programas parciais são construídos começando no instante 0 e são sistematicamente formados ao longo do processo de busca ao adicionar, a cada ponto de decisão, um subconjunto de atividades até que um programa completo e factível seja obtido. Neste sentido, um programa completo é uma continuação de um programa parcial.

A cada instante de tempo t, define-se um conjunto elegível E_t, como o conjunto das atividades que não estão no programa parcial e cujos predecessores terminaram de ser processados. Estas atividades elegíveis podem começar no instante t, se as restrições de recursos não forem violadas. Os próximos dois teoremas descrevem dois casos especias explorados pelo algoritmo para colocar uma atividade elegível em progresso.

Teorema 1. Se em qualquer instante de tempo t, o programa parcial PS_t não tem nenhuma atividade em progresso e uma atividade elegível i não pode ser programada junto com qualquer outra atividade que ainda não foi programada em qualquer instante de tempo $t' \geq t$, sem violar as restrições de precedência ou recurso, então existe uma continuação ótima do programa parcial com a atividade elegível i colocada em progresso no instante t.

Teorema 2 Se no instante de tempo t o programa parcial PS_t não tem atividades em progresso, se existe uma atividade i elegível que pode ser programada simultaneamente com apenas uma outra atividade não programada j, em qualquer instante de tempo $t' \geq t$, sem violar as relações de precedência ou recursos, e se a atividade j é elegível e tem uma duração menor ou igual do que a atividade i, então existe uma continuação ótima do programa parcial em que ambas atividades i e j são colocadas em progresso no instante t.

A cada ponto de decisão t, este teorema permite que o procedimento identifique as duas atividades elegíveis que devem ser programadas simultaneamente.

Se for impossível programar todas as atividades elegíveis no instante t, diz-se que um conflito ocorre. Tal conflito produzirá um nova ramificação na árvore do *branch-and-bound*. Os ramos descrevem modos de resolver o conflito de recursos, isto é, eles correspondem a decisões sobre quais combinações de atividades devem ser atrasadas. Para isso, é definido um conjunto de atraso $D(p)$, que consiste de todos os subconjuntos de atividades D_q que estão em progresso ou elegíveis, cujo atraso resolveria o conflito corrente de recursos no nível p da árvore. Uma alternativa de atraso D_q é mínima se ela não contém outras alternativas de atraso $D_v \in D(p)$ como um subconjunto.

O próximo teorema permite ao procedimento eliminar alternativas de atraso que são redundantes. O atraso de um subconjunto de atividades $D_q \in D(p)$ é introduzido no método, através da adição de novas relações de precedência. Estas novas relações de precedência obrigam as atividades a esperar o término de outras atividades, para serem processadas. Para cada alternativa de atraso $D_q \in D(p)$ o conjunto de relações de precedência adicionais $G_q = \{(j,i)\}$ é construído tomando, como predecessor de todas as atividades $i \in D_q \in D(p)$, a atividade j com instante de término mais cedo, que se encontra em progresso ou é elegível para ser processada no instante t, e que não está atrasada (em caso de empate, a seleção da atividade é arbitrária).

Teorema 3. Para resolver um conflito de recurso, é suficiente considerar somente alternativas de atrasos mínimas.

Cada alternativa de atraso D_q será avaliada através de um limitante inferior baseado em caminho crítico e denotado limitante inferior crítico L_q. Este limitante, que considera simultaneamente tanto restrições de precedência como recursos, é descrito a seguir. Considere a alternativa de atraso D_q, para a qual deseja-se determinar o limitante inferior crítico da seqüência. Utilizando as relações de precedência adicionais G_q, determina-se o programa parcial correspondente *PS'* (isto é, $PS_t+E_t-D_q$) e realizam-se os cálculos de caminho crítico, baseado em relações de precedência, para o grafo. Considere um caminho crítico (empates são resolvidos arbitrariamente) com tamanho z. Todas as atividades não-programadas, $i \notin PS'$, que não estão

neste caminho, são inseridas no conjunto *NP*. O cálculo do instante de início mais cedo s_i e instante de término mais tarde lf_i das atividades de *NP* é baseado somente em relações de precedência. Se existe uma atividade não-programada $i \in NP$ de duração d_i e se recursos de um ou mais tipos, só estão disponíveis em quantidade suficiente para serem alocados para atividade *i* continuamente por $e_i \leq d_i$ períodos de tempo, em algum lugar no intervalo de e_i a lf_i, então o término do projeto será atrasado por no mínimo d_i-e_i períodos de tempo. O limitante inferior da seqüência crítica pode então ser calculado como $L_q = \max_{i \in NC}\{z + d_i - e_i\}$. A ramificação continua a partir do nó correspondente à alternativa de atraso D_q^* de menor valor de L_q^*.

Duas regras de dominância são utilizadas para podar a árvore. A primeira regra é aplicada sempre que for possível mostrar que uma atividade pode ser deslocada para a esquerda e o programa parcial resultante é tanto factível do ponto de vista de precedência como de recurso. Esta regra é implementada da seguinte forma: defina um conjunto $DS = \{j \in D_q^* \mid f_q < t + d_j\}$ como o conjunto das atividades que começaram antes do instante *t* mas que precisam ser atrasadas quando a alternativa de atraso D_q^* é selecionada. Se *DS* não é vazio, esta regra de dominância é invocada utilizando a seguinte estrutura de seleção: se as relações de precedência que foram adicionada em níveis anteriores da árvore de busca forçam a atividade *i* a se tornar elegível no instante *t*, se a decisão corrente foi começar aquela atividade no instante *t*, e se o conjunto de atraso de atividades *DS* permite que a atividade *i* possa ser deslocada para a esquerda sem causar um conflito de recursos, então o programa parcial correspondente é dominado.

A segunda regra de dominância é baseada no conceito de um conjunto de corte. Em todo instante de tempo *t*, um conjunto de corte C_t é definido como o conjunto de todas as atividades ainda não programadas, para as quais, todos os predecessores pertencem ao programa parcial PS_t.

Teorema 4. Considere um conjunto de corte *(cutset)* C_t no instante *t* que contém as mesmas atividades que o conjunto de corte $C_{t'}$, que foi guardado anteriormente durante a busca de um outro caminho na árvore. Se o instante *t'* não for maior do que o instante *t* e se todas as atividades em progresso no instante *t'* não terminam mais tarde do que o máximo de *t* e o instante de término das atividades correspondentes em PS_t, então o programa parcial corrente PS_t é dominado.

Backtraking ocorre quando um programa é completado ou um ramo é podado pelo cálculo do limitante inferior e/ou regra de dominância. No *backtrack* os arcos adicionados correspondentes ao último elemento do conjunto de atraso $D_q^* \in D(p)$ são removidos e novos arcos são adicionados para o próximo elemento no mesmo nível. Se não existe nenhuma alternativa de atraso D_q inexplorada neste nível, o procedimento realiza um *backtrack* para o nível anterior. Quando o nível zero da árvore de busca é atingido, o processo de busca está completo e a solução ótima foi encontrada e verificada.

O algoritmo branch-and-bound

A seguir, o algoritmo *branch-and-bound* é descrito em detalhes. Para simplicidade de notação, os autores omitem o subscrito *t* dos conjuntos *PS*, *S*, *E* e *C*. A operação *save* realizada no Passo 2 abaixo deve ser distinta da operação *store* realizada no Passo 5. A operação *save* guarda a informação do conjunto de corte necessária para aplicar a regra de dominâcia do conjunto de corte. A operação *store* guarda informação que é recuperada durante o *backtracking*.

Passo 1. Seja $T = 9999$ um limitante superior para a duração do projeto. Atribua, para o nível da árvore de *branch-and-bound*, o valor $p = 0$. Inicialize $t = 0$. Para cada atividade i, calcule o restante do caminho crítico $RCPL_i$. Inicialize os instantes de término das atividades $f_i = 9999$. Programe a atividade artificial inicial, isto é, faça $f_1 = 0$, e atualize o programa parcial $PS = \{1\}$ e o conjunto das atividades em progresso $S = \{1\}$. Calcule o limitante inferior: $LB(0) = RCPL_i$. Atualize o conjunto de corte: $C = \{x \mid x$ tem atividade 1 como único predecessor$\}$ e atribua o valor zero aos tempos de início mais cedo das atividades do conjunto de corte s_x.

Passo 2. Calcule o próximo ponto de decisão t, dado pelo instante de término mais cedo de todas as atividades em progresso: $t = \min\{f_i, i \in S\}$. Para todas as atividades $j \in S$ tais que $f_j = t$, atualize o conjunto das atividades em progresso: $S = S - \{j\}$.

Se a última atividade programada é a atividade artificial n, o programa está completo. Atualize o *makespan* do projeto $T = f_n$. Se T é igual a $LB(0)$, então pare (com a solução ótima), caso contrário vá para o Passo 7 (*backtrack*).

Verifique se o conjunto de corte corrente C é dominado por um conjunto de corte previamente guardado. Se isto acontecer, vá para o Passo 7 (*backtrack*), caso contrário guarde o conjunto de corte corrente C, guarde o conjunto S das atividades em progresso junto com seus instantes de término f_j e guarde também o ponto de decisão corrente t.

Construa o conjunto das atividades elegíveis: $E = \varnothing$ e para cada atividade $i \in C$ com $s_i = t$ atualize o conjunto elegível $E = E \cup \{i\}$. Se não existirem atividades elegíveis (isto é, se $E = \varnothing$), vá para o Passo 2. Se existem ainda algumas atividades em progresso, vá para o Passo 4, caso contrário, continue com o Passo 3.

Passo 3. Para cada atividade elegível i $\in E$ conte o número de atividades não programadas j (não necessariamente elementos de E) que podem ser processadas simultaneamente com a atividade i sem violar as relações de precedência e recursos. Se nenhuma pode ser processada simultaneamente com a atividade elegível i, então coloque a atividade elegível em progresso: $PS = PS \cup \{i\}$, $S = \{i\}$, $f_i = t + d_i$ e atualize o conjunto $C = C\text{-}\{i\}+\{x \mid x$ é um sucessor imediato de i com todos os seus predecessores em $PS\}$. Para todas as atividades do conjunto de corte, $x \in C$, atualize os tempos de início mais cedo $s_x = f_i$.

Se a atividade elegível i pode ser programada simultaneamente com somente uma outra atividade j, que é elegível e não tem uma duração maior do que a atividade i, coloque ambas atividades i e j em progresso e atualize o conjunto de corte: $C = C\text{-}\{i, j\} + \{x \mid x$ é um sucessor imediato de i ou j, com todos os seus predecessores no programa parcial$\}$. Para todo $x \in C$ atualize os tempos de início mais cedo $s_x = f_i$.

Se uma atividade elegível foi programada durante este passo, vá para o Passo 2, caso contrário vá para o Passo 4.

Passo 4. Temporariamente coloque todas as atividades elegíveis em progresso: $PS = PS \cup E$, $S = S \cup E$, faça $f_i = t + d_i$ para todo $i \in E$. Atualize o conjunto de corte: $C = C - E + \{x \mid x$ é um

sucessor imediato de $i \in E$, com todos os predecessores em PS} e para todas as atividades do conjunto de corte, atribua o valor $s_x = \max\{f_a \mid (a,x) \in H\}$ aos tempos de início mais cedo.

Para cada tipo de recurso k, verifique se $\sum_{i \in S} r_{ik} \leq a_k$. Se existe ao menos um recurso do tipo k para o qual a soma dos recursos de todas as atividades em progresso excedem a disponibilidade de recursos, então um conflito de recurso ocorre: vá para o Passo 5, caso contrário vá para o Passo 2.

Passo 5. Atualize o nível de ramificação na árvore de busca: $p = p + 1$. Determine, para cada tipo de recurso k, quantas unidades de recurso precisam ser liberadas para resolver o conflito de recurso, isto é, para cada k faça $w_k = \sum_{i \in S} r_{ik} - a_k$. Defina o conjunto de atraso $D(p)$ = { $D_q \subseteq S \mid \sum_{i \in D_q} r_{ik} \geq w_k$ para todo k e D_q que não contenha outro $D_v \in D(p)$ como um subconjunto}.

Para cada $D_q \in D(p)$ determine a atividade j com instante de término mais cedo, que esteja em progresso e que não está atrasada (isto é, $j \in (S - D_q)$). Define o conjunto correspondente de duplas $G_q = \{(j, i)\}$ para todo $i \in D_q$. Calcule, para cada alternativa de atraso D_q, o limitante de seqüência crítica L_q.

Selecione o $D_q^* \in D(p)$ com o menor L_q^*. Atualize o conjunto de atraso: $D(p) = D(p) - D_q^*$. Faça $LB(p) = \max\{LB(p\text{-}1), L_q^*\}$. Se $LB(p) \geq T$, vá para o Passo 7. Caso contrário, guarde (*store*) o instante de término das atividades, o programa parcial, o conjunto de atividades em progresso, as atividades do conjunto de corte com seus tempos de início mais cedo e o ponto de decisão t.

Passo 6. (Gere um novo nó). Defina DS como o conjunto de todas as atividades que começaram mais cedo do que t mas que precisam ser atrasadas: $DS = \{i \in D_q^* \mid f_i < t + d_i\}$. Adicione as relações de precedência adicionais neste nível: $H = H \cup G_q^*$. Atualize $PS = PS - D_q^*$, $S = S - D_q^*$. Para todo $i \in D_q^*$ faça f_i igual a 9999. Atualize o conjunto de corte: $C = C + D_q^*$ - $\{r \mid x \in D_q^*$ e $(x, r) \in H\}$. Para $i \in D_q^*$, atribua os seguintes valores aos tempos de início mais cedo: $s_i = \{f_j \mid (j, i) \in G_q^*\}$.

Se *DS* não está vazio, aplique a regra de dominância de deslocamento à esquerda da seguinte forma. Se as relações de precedência que foram adicionadas em níveis anteriores da árvore de busca forçaram uma atividade *i* a se tornar elegível no instante *t*, se a decisão corrente fosse começar aquela atividade no instante *t* e se o conjunto de atraso de atividades *DS* permite que esta atividade *i* possa ser deslocada à esquerda sem causar um conflito de recursos, então este programa é dominado: vá para o Passo 7 (*backtrack*); caso contrário, vá para o Passo 2.

Passo 7. Se o nível da árvore for $p = 0$, então pare. Caso contrário, elimine as relações de precedência que foram adicionadas neste nível: $H = H - G_q^*$. Se $D(p) = \varnothing$, faça $p = p - 1$ e repita o Passo 7.

Selecione o $D_q^* \in D(p)$ com o menor L_q^*. Atualize o conjunto de atraso: $D(p) = D(p) - D_q^*$. Calcule o limitante inferior $LB(p) = \max\{LB(p - 1),\ L_q^*\}$. Se $LB(p) \geq T$, decresça o nível de ramifique: $p = p - 1$ e repita o Passo 7.

Recupere os instantes de término das atividades, o programa parcial, o conjunto de atividades em progresso, o conjunto de corte das atividades, os tempos de início mais cedo das atividades do conjunto de corte e o ponto de decisão *t*. Vá para o Passo 6.

Exemplo numérico

O seguinte exemplo foi descrito no artigo de Demeulemeester e Herroelen (1992). Considere o grafo de relações de precedência da Figura 2. Os números acima de cada nó denotam a duração das atividades. Neste exemplo, existe apenas um único tipo de recurso, e o número abaixo de cada nó denota a quantidade de recurso requerida pela atividade. O recurso tem uma disponibilidade constante de 6 unidades.

Ao iniciar o *branch-and-bound*, faça $T = 9999$ e calcule a extensão do caminho crítico, $RCPL_i = 10$. Inicialize o nível da árvore de *branch-and-bound* e o ponto de decisão: $p = 0$, $t = 0$. Inicialize todos os instantes de término das atividades $f_i = 9999$. Programe a atividade artificial 1: $f_1 = 0$. O programa parcial é atualizado para $PS = \{1\}$ e o conjunto das atividades em progresso é dado por $S = \{1\}$. O limitante inferior é calculado como $LB(0) = 10$. O conjunto de corte é

calculado como $C = \{2, 3, 4, 5\}$ com $s_2 = s_3 = s_4 = s_5 = 0$. Isto corresponde ao nó 1 no nível 0 da árvore de busca ilustrada na Figura 3.

Prosseguindo com o Passo 2, o instante de término mais cedo de todas as atividades em progresso é $t = 0$, que é o próximo ponto de decisão. Atualize $S = \varnothing$. O conjunto de corte corrente $C = \{2, 3, 4, 5\}$ não é dominado, e portanto deve ser guardado (*saved*). O conjunto das atividades elegíveis é calculado como $E = \{2, 3, 4, 5\}$.

No Passo 3, é possível verificar que não existem atividades em E que devem ser programadas sozinhas (Teorema 1), assim como não existe uma atividade elegível que possa ser programada simultaneamente com apenas uma outra atividade que é tanto elegível como menor em duração (Teorema 2). Prossegue-se portanto, para o Passo 4. As atividades elegíveis são colocadas, temporariamente, em progresso: $PS = \{1, 2, 3, 4, 5\}$, $S = \{2, 3, 4, 5\}$, $f_2 = 2$, $f_3 = 3$, $f_4 = 5$ e $f_5 = 4$. Atualize o conjunto de corte $C = \{6, 8\}$, $s_6 = 2$ e $s_8 = 3$.

A utilização de recursos do tipo 1, somado sobre todas as atividades em progresso, é igual a 8 unidades. Um conflito de recurso ocorre, portanto, prossegue-se para o Passo 5. Faça $p = 1$ (nível da árvore de busca na Figura 3) e calcule o número de unidades de recurso que devem ser liberados para resolver o conflito de recursos: $w_1 = 8 - 6 = 2$. O conjunto de atraso mínimo é $D(1)=\{\{3\},\{4\},\{5\}\}$. A alternativa {2,3}, por exemplo, poderia ser uma alternativa válida de atraso, mas não é considerada pois o subconjunto {3} já pertence ao conjunto de atraso. Para a alternativa de atraso $D_1 = \{3\}$, a atividade 2 é a atividade com instante de término mais cedo, que não está atrasada. Faça $G_1 = \{2, 3\}$ e calcule o limitante inferior L_1 da seguinte forma: considere a relação de precedência (2,3) que foi adicionada, o correspondente programa parcial resultante é $PS' = \{1, 2, 4, 5\}$. O caminho crítico obtido é 1-2-3-8-9 com um tamanho de $z = 11$ unidades de tempo. Isto dá origem à seqüência crítica 2-3-8 como indicado na Figura 4. Considere agora a atividade 6, ainda não programada, com $es_6 = 2$ e $lf_6 = 5$, que necessita de 2 unidades de recurso. Esta atividade pode ser programada continuamente por um período $e_6 = d_6 = 2$ unidades de tempo no intervalo de 2 até 5. Do mesmo modo, a outra atividade não programada 7, com $es_7 = 4$ e $lf_7 = 11$ e que precisa de 1 unidade de recurso pode ser programada continuamente por um período de $e_7 = d_7 = 6$ unidades de tempo. Portanto, o limitante inferior de seqüência crítica é calculado para estas alternativas de atraso como $L_1 = \max\{(z + d_6 - e_6), (z + d_7 - e_7)\} = \max(11 + 2 - 2), (11 + 6 - 6)\} = 11$.

Prosseguindo de modo semelhante, para $D_2 = \{4\}$ obtém-se a dupla $G_2 = \{(2, 4)\}$ com $L_2 = 10$. Para a alternativa de atraso $D_3 = \{5\}$, encontra-se $G_3 = \{(2,5)\}$ com $L_3 = 10$.

Selecione $D_3^* = \{5\}$, que possui a menor estimativa de limitante inferior $L_3^* = 10$ (empates são resolvidos arbitrariamente). Atualize o conjunto de atraso $D(1) = \{\{3\},\{4\}\}$. Calcule o limitante inferior, $LB(1) = \max\{LB(0), L_3^*\} = \max(10, 10) = 10$. Guarde (*store*) o programa parcial $PS = \{1, 2, 3, 4, 5\}$, o conjunto das atividades em progresso $S = \{2, 3, 4, 5\}$, os instantes de término das atividades $f_2 = 2$, $f_3 = 3$, $f_4 = 5$ e $f_5 = 4$, o conjunto de corte $C = \{6, 8\}$, o instante de início mais cedo das atividades do conjunto de corte $s_6 = 2$, $s_8 = 3$ e o ponto de decisão $t = 0$.

Prossiga com o Passo 6 e o nó 2 da árvore. Adicione a nova relação de precedência (2, 5). Atualize $PS = \{1, 2, 3, 4\}$, $S = \{2, 3, 4\}$, $f_5 = 9999$. Atualize o conjunto de corte: $C = \{5, 6, 8\}$ com $s_5 = 2$, $s_6 = 2$, $s_8 = 3$. Como $DS = \varnothing$, a regra de dominância de deslocamento à esquerda não se aplica e o procedimento continua com o Passo 2.

O instante de término mais cedo de todas as atividade em progresso é $t = 2$, que será o próximo ponto de decisão. Atualize o conjunto das atividades em progresso $S = \{3, 4\}$. O conjunto de corte corrente é dado por $C = \{5, 6, 8\}$. Como não existe nenhum conjunto de corte idêntico, guardado anteriormente, a regra de dominância do conjunto de corte não se aplica. Guarde (*save*) o conjunto de corte $C = \{5, 6, 8\}$, o conjunto de atividades em progresso $S = \{3, 4\}$, os instantes de término $f_3 = 3$, $f_4 = 5$ e o ponto de decisão $t = 2$. O conjunto das atividades elegíveis é atualizado: $E = \{5, 6\}$. As atividades 3 e 4 ainda estão em progresso, portanto, prosseguimos com o Passo 4.

As atividades elegíveis são postas temporariamente em progresso: $S = \{3, 4, 5, 6\}$, $f_5 = 6$, $f_6 = 4$, $PS = \{1, 2, 3, 4, 5, 6\}$. Atualize o conjunto de corte: $C = \{7, 8\}$ e os instantes de início mais cedo das atividades pertencentes ao conjunto de corte: $s_7 = 4$, $s_8 = 3$. A soma das necessidades de recursos de todas atividades em progresso é igual a 9 unidades. Um conflito de recurso ocorre. Prossiga com o Passo 5.

Faça $p = 2$ (nível da árvore de busca da Figura 3) e calcule a quantidade de recursos que deve ser liberada para resolver o conflito: $w_1 = 9 - 6 = 3$. O conjunto de atraso é dado por $D(2) = \{\{4\},\{3,5\},\{3,6\},\{5,6\}\}$. Para a alternativa de atraso $D_1 = \{4\}$, temos que $G_1 = \{(3, 4)\}$ com $L_1 = 10$. Para a segunda alternativa de atraso $D_2 = \{3, 5\}$, temos que a atividade 6 é o predecessor,

gerando G_2={(6,3),(6,5)} com L_2 = 13. De forma similar encontra-se G_3 = {(4,3),(4,6)}, com L_3 = 14 e G_4 = {(3,5), (3,6)}com L_4 = 11.

A menor estimativa de limitante inferior é dada por D_1^* = {4} com L_1^* = 10, portanto, esta alternativa de atraso é selecionada. Atualize o conjunto de atrasos $D(2)$ = {(3,5), (3,6), (5,6)}. Calcule o limitante inferior como $LB(2)$ = max(10, 10) = 10 < T. Armazene S = {3, 4, 5, 6}, f_3 = 3, f_4 = 5, f_5 = 6, f_6 = 4, PS = {1, 2, 3, 4, 5, 6}, C = {7, 8}, s_7 = 4, s_8 = 3 e o ponto de decisão t = 2. Prossiga com o Passo 6 e crie o nó 3 da árvore de busca. Atualize PS = {1, 2, 3, 5, 6}, S = {3, 5, 6}. Inclua as relações de precedência adicionais (3, 4), f_4 = 9999. Atualize o conjunto de corte C = {4, 7, 8} com s_4 = 3, s_7 = 4 e s_8 = 3. Como DS = {4}, faz-se uma tentativa de aplicar a regra de dominância de deslocamento à esquerda. A atividade 5 foi atrasada anteriormente pela relação de precedência adicional (2,5) e se tornou elegível novamente no instante t = 2. A decisão corrente é iniciar a atividade 5 no instante 2. Como indicado na Figura 5, a decisão corrente de atrasar a atividade 4, que estava em progresso no programa pai, permite a atividade 5 ser deslocada à esquerda e começar no instante 0. O nó 3 é dominado e prossegue-se com o Passo 7 do procedimento (*backtrack*). A relação de precedência (3, 4) é eliminada. Seleciona-se D_4^* = {5, 6} do conjunto de atraso $D(2)$ = {{3,5},{3,6}{5,6}}, pois esta é a alternativa de atraso com o menor valor de L_4^* = 11. Atualize o conjunto de atraso $D(2)$ = {{3,5},{3,6}} e calcule o limitante inferior como $LB(2)$ = max(10, 11) = 11 < T. Recupere S = {3, 4, 5, 6}, f_3 = 3, f_4 = 5, f_5 = 6, f_6 = 4, PS = {1, 2, 3, 4, 5, 6}, C = {7, 8}, s_7 = 4, s_8 = 3 e o ponto de decisão t = 2. Vá para o Passo 6.

Crie o nó 4 da árvore de busca. Atualize o programa parcial PS = {1, 2, 3, 4} e o conjunto das atividades em progresso S = {3, 4}. Adicione as relações de precedência adicionais (3,5) e (3,6): f_5 = f_6 = 9999. Atualize o conjunto de corte C = {5, 6, 8} com s_5 = 3 e s_6 = 3 e s_8 = 3. Como DS está vazio, a regra de dominância de deslocamento à esquerda não se aplica. Continue com o Passo 2.

Faça t = 3, que corresponde ao término da atividade 3. Atualize S = {4}. O conjunto de corte corrente C = {5, 6, 8} é idêntico a um dos conjuntos de cortes já salvos no instante de tempo anterior t = 2. Entretanto, a regra de dominância não se aplica porque este conjunto salvo anteriormente foi gerado no nó pai 2 no mesmo caminho da árvore de busca. Guarde o conjunto de corte C = {5, 6, 8}, o conjunto das atividades em progresso S = {4} com f_4 = 5 e o ponto de decisão t = 3. O conjunto elegível é agora dado por E = {5, 6, 8}. Temporariamente coloque todas

as atividades elegíveis em progresso: $PS = \{1, 2, 3, 4, 5, 6, 8\}$, $S = \{4, 5, 6, 8\}$ com $f_4 = 5$, $f_5 = 7$, $f_6 = 5$ e $f_8 = 9$. Atualize o conjunto de corte: $C = \{7\}$ com $s_7 = 5$. Um conflito de recursos ocorre novamente. Prossiga com o Passo 5, faça $p = 3$ e gere $D(3) = \{\{4\},\{5\},\{6\}\}$. Temos então $G_1 = \{(6, 4)\}$ com $L_1 = 11$; $G_2 = \{(4, 5)\}$ com $L_2 = 11$ e $G_3 = \{(4, 6)\}$ com $L_3 = 13$. Seleciona-se então a alternativa de atraso D_2^* com $L_2^* = 11$ e calcule o limitante inferior $LB(3) = \max(11, 11) = 11$. Guarde (*store*) o conjunto $S = \{4, 5, 6, 8\}$ com $f_4 = 5$, $f_5 = 7$, $f_6 = 5$ e $f_8 = 9$, o programa parcial $PS = \{1, 2, 3, 4, 5, 6, 8\}$, o conjunto de corte $C = \{7\}$ com $s_7 = 5$ e o ponto de decisão $t = 3$.

Crie o nó 5 da árvore de busca. Atualize $PS = \{1, 2, 3, 4, 6, 8\}$, $S = \{4, 6, 8\}$. Adicione a relação de precedência extra (4,5): $f_5 = 9999$. Atualize o conjunto de corte: $C = \{5,7\}$ com $s_5 = 5$ e $s_7 = 5$. Como $DS = \varnothing$, a regra de dominância do deslocamento à esquerda não se aplica e prossegue-se para o Passo 2.

O próximo ponto de decisão agora ocorre no instante $t = 5$, marcado pelo término das atividades 4 e 6. Atualize $S = \{8\}$. O conjunto de corte corrente não é dominado. Guarde (*save*) o conjunto de corte $C = \{5,7\}$ com $s_5 = 5$ e $s_7 = 5$ e as atividades em progresso $S = \{8\}$ com $f_8 = 9$. As atividades 5 e 7 são elegíveis. Atividades 5 e 7 são temporariamente colocadas em progresso: $PS = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $S = \{5, 7, 8\}$ com $f_5 = 9$, $f_7 = 11$ e $f_8 = 9$. Atualize $C = \{9\}$ com $s_9 = 11$. Nenhum conflito de recurso ocorre. O próximo ponto de decisão é $t = 9$, marcado pelo término das atividades 5 e 8: $S = \{7\}$. O conjunto de corte corrente não é dominado. Guarde (*save*) $C = \{9\}$ com $s_9 = 11$, $S = \{7\}$ com $f_7 = 11$ e $t = 9$. Nenhuma atividade se torna elegível. O próximo ponto de decisão ocorre no instante $t = 11$ quando a atividade 7 é completada: $S = \varnothing$. O conjunto de corte corrente não é dominado uma vez que existe somente um conjunto de corte idêntico, que foi obtido no mesmo caminho. Guarde $C = \{9\}$, $S = \varnothing$ e $t = 11$. $E = \{9\}$. Coloque a atividade 9 em progresso: $PS = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $f_9 = 11$, $C = \varnothing$. Foi encontrada uma solução factível completa com $T = 11$.

Continuando com o Passo 7 a relação de precedência é eliminada e seleciona-se a alternativa de atraso $D_1^* = \{4\}$ com $L_1^* = 11$. O limitante inferior correpondente é calculado: $LB(3) = \max(11,11) = 11 = T$. Elimine o nó (o nó correpondente à relacão de precedência (6,4) está indicado em linhas pontilhadas na Figura 3). Como sabemos com certeza que a alternativa de atraso restante D_3 não pode ter uma estimativa melhor para o limitante inferior, o nível de ramificação passa a ser $p = 2$ (nível 2 da árvore de busca) e o Passo 7 é repetido.

As relações de precedência (3,5) e (3,6) que foram adicionadas no nó 4 são eliminadas. Selecione $D_2 = \{3,5\}$ com $L_2^* = 13$. Calcule o limitante inferior $LB(2) = \max(10,13) = 13 > T$. Elimine o nó. As alternativas de atraso restantes não podem ter uma estimativa melhor para o limitante inferior: faça $p = 1$ e *backtrack* para o nível 1 da árvore. Elimine as relações de precedência (2,5) e selecione $D_2^* = \{4\}$ com $L_2^* = 10$. Calcule o limitante inferior $LB(1) = \max(10,10) = 10 < T$. Recupere os valores e prossiga com o Passo 6.

O procedimento prosseguirá a exploração a árvore como indicado na Figura 3, atingindo uma solução ótima com *makespan* igual a 10 unidades de tempo e $f_1 = 0, f_2 = 2, f_3 = 3, f_4 = 9, f_5 = 4, f_6 = 4, f_7 = 10, f_8 = 9$ e $f_9 = 10$.

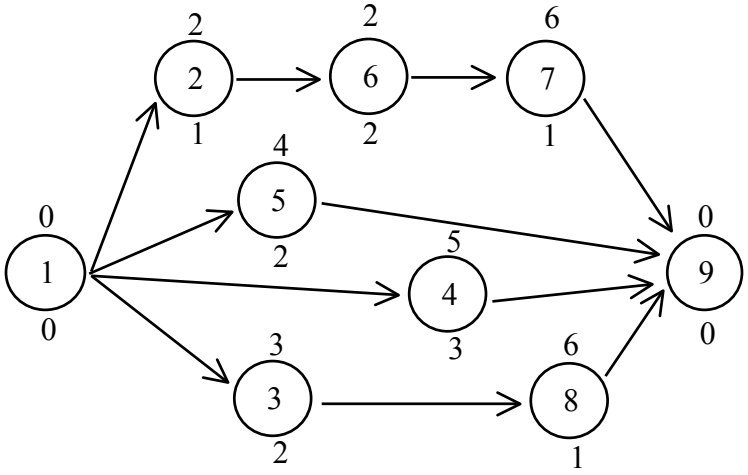

Figura 2. Relações de precedência para o exemplo do PPPRL

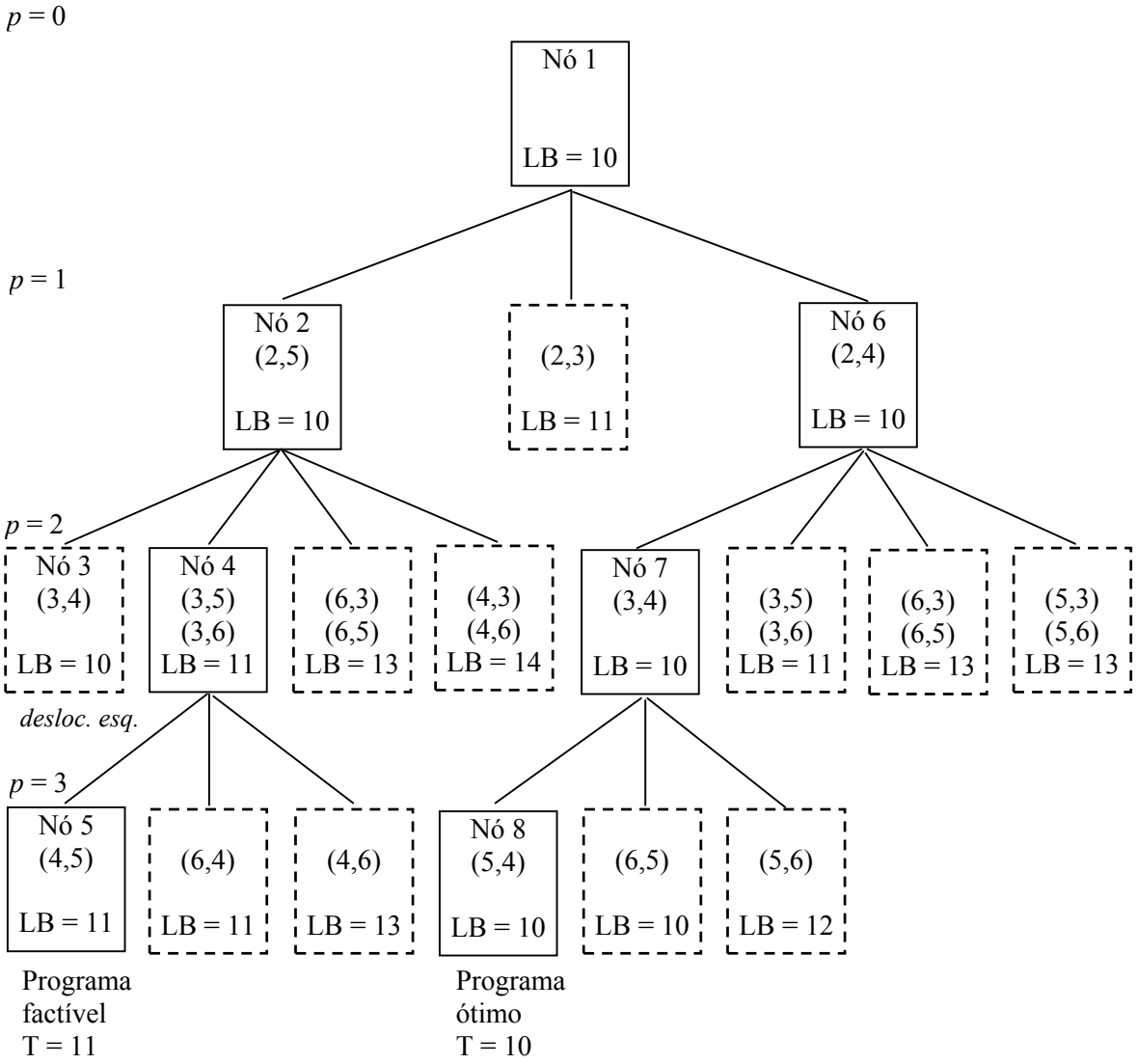

Figura 3. Árvore de busca do *branch-and-bound.*

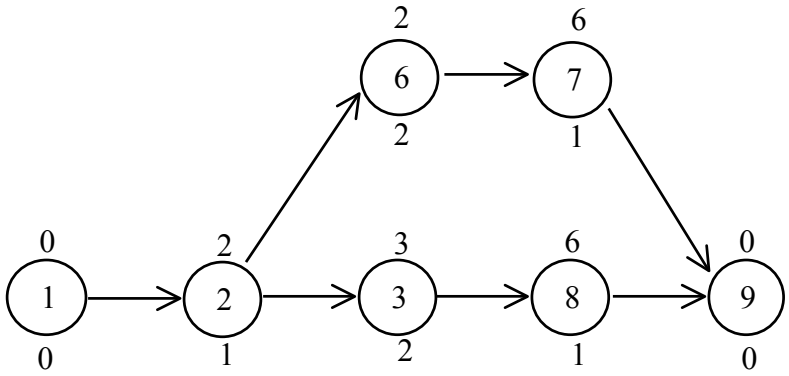

Figura 4a. Ilustração do limitante de sequência crítica – Grafo do projeto

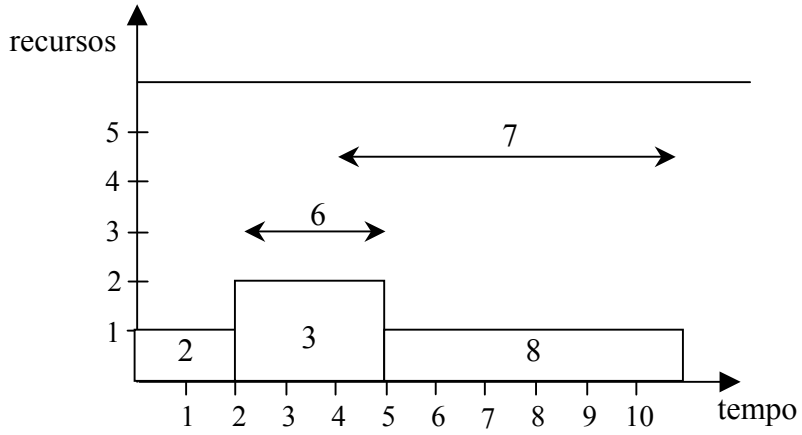

Figura 4b. Ilustração do limitante de sequência crítica - Atividades não-programadas 6 e 7 podem ser programadas

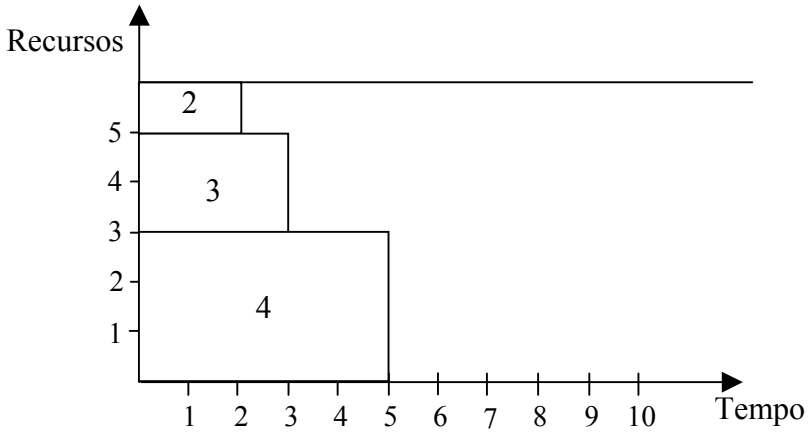

Figura 5a. Ilustração da regra de dominância do deslocamento à esquerda – Programa parcial *PS* = {1,2,3,4} para o Nó 2 da árvore

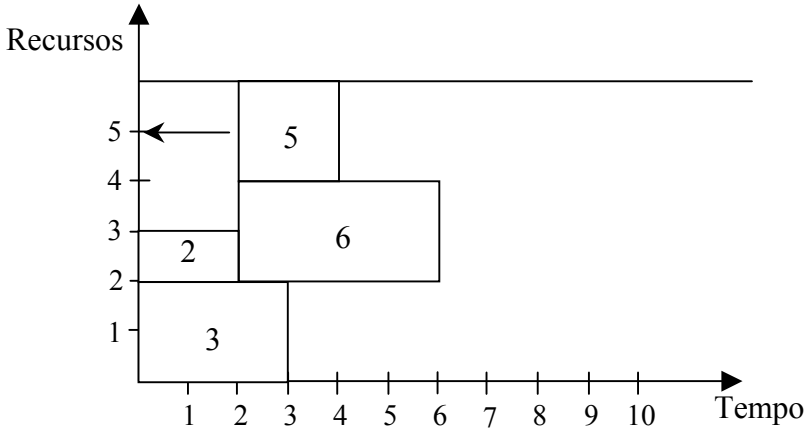

Figura 5b. Ilustração da regra de dominância do deslocamento à esquerda - Programa parcial dominado *PS* = {1,2,3,5,6}