PROBLEMA DE CUSTO DE UTILIZAÇÃO DE RECURSOS EM PROGRAMAÇÃO DE PROJETOS

Denise Sato Yamashita
Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas
CP 6101, CEP 130 3- 70, Campinas, SP, Brasil, denise@densis.fee.unicamp.br

Vinícius Amaral Armentano
Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas
CP 6101, CEP 130 3- 70, Campinas, SP, Brasil, vinicius@densis.fee.unicamp.br

Resumo

Este trabalho aborda o problema de programação de projetos no qual deseja-se minimizar os custos de utilização de recursos, levando em consideração uma data de entrega para o projeto e relações de precedência entre as atividades. Heurísticas baseadas nos métodos *scatter search* e *multi-start* são propostas. Resultados computacionais são apresentados para 432 problemas teste.

Palavras-chave: programação de projetos, custo de utilização de recursos, heurísticas

Abstract

This paper considers the project scheduling problem where the goal is to minimize the resource availability costs taking into account a due date for the project and precedence relations among the activities. Heuristics based on the scatter search algorithm and multi-start method are proposed. Computational results are reported for 432 instances.

Key-words: project scheduling, resource availability cost, heuristics

1. Introdução

Um projeto consiste de diversas atividades que requerem tempo e recursos limitados para sua execução. Associado ao projeto, existe também um objetivo que deseja-se atingir, como por exemplo, minimização do tempo de duração do projeto ou variação do uso de recursos. Além disso, podem existir relações de precedência entre as atividades, indicando a ordem em que estas devem ser realizadas. A programação de projetos, em sua forma mais básica, consiste em encontrar tempos de início para todas atividades tal que recursos e relações de precedência sejam satisfeitos e uma função objetivo seja minimizada.

Os problemas de programação de projetos aparecem em várias situações práticas, como desenvolvimento de novos produtos, planejamento de produção no setor de manufatura, construção civil (Neumann *et al*., 2002). Desde o final dos anos 50, diversos métodos de solução para problemas de programação de projetos foram desenvolvidos (Moder *et al*., 1 3). Um problema importante de programação de projetos é o *problema de programação de projetos com recursos limitados* (PPPRL) no qual deseja-se minimizar o *makespan* sujeito a recursos limitados. Este trabalho trata da situação na qual o custo de utilização dos recursos e data de entrega para o projeto são levados em consideração. Este problema é conhecido como *problema de minimização do custo de utilização de recursos* (PCUR).

O objetivo deste artigo é desenvolver uma heurística para o PCUR. Dois métodos de resolução são propostos, *scatter search* e *multi-start*. A seção 2 apresenta uma descrição do PCUR, a seção 3 discute o procedimento de avaliação de solução e descreve a heurística de melhoria utilizada tanto no método *scatter search* como no método *multi-start*, a seção 4 apresenta a heurística *multi-start*, a seção 5 descreve a heurística *scatter search*, a seção 6 apresenta os resultados computacionais e a seção 7 apresenta as conclusões deste trabalho.

2. Problema de custo de utilização de recursos (PCUR)

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, respectivamente. Cada atividade i tem duração d_i e requer r_{ik} unidades de recurso do tipo k (k= 1, ..., m) para ser executada. 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 alocação de recursos a_k. O tempo de término da atividade i é representado por f_i, e a_k é a quantidade de recursos do tipo k alocada. Sejam $K = (a_1, ..., a_m)$ e $d = (d_1,...,d_n)$, o vetor de alocação de recursos e o vetor de duração das atividades, respectivamente. Um modelo conceitual do problema de custo de utilização de recurso é apresentado a seguir.

Minimizar $\sum_k C_k(a_k)$ (1)
Sujeito a

$$f_j \geq f_i + d_{js} \qquad \forall (i, j) \in H \qquad (2)$$

$$f_1 = 0 \qquad (3)$$

$$f_n \leq D \qquad (4)$$

$$\sum_{A_t} r_{ik} \leq a_k \qquad k = 1,..., m \quad \text{e} \quad t = 1,...., f_n \qquad (5)$$

O conjunto A_t, na formulação acima, representa o conjunto das atividades em progresso durante o intervalo (t-1,t]. As variáveis de decisão do problema são o tempo de término da atividade i, f_i, e a quantidade a_k de recursos do tipo k alocada. É interessante notar a relação do PCUR com o problema de programação de projetos com recursos limitados (PPPRL). A formulação do PPPRL é semelhante ao PCUR: a restrição (4) da formulação acima é retirada e a função objetivo é modificada para minimizar f_n. O problema de programação de projetos com recursos limitados é NP-hard (Blazewicz *et al*., 1 3) e tem sido estudado extensivamente na literatura. Revisões bibliográficas recentes sobre métodos exatos e heurísticos podem ser encontradas em Herroelen *et al.* (1), Brucker *et al.* (1) e Kolisch e Hartman (1).

A literatura para o PCUR é bastante restrita. Möhring (1 4) propõe um método de solução baseado num procedimento desenvolvido por Radermacher (1 5) para o PPPRL, e aplica o algoritmo a um problema prático de construção de uma ponte. Demeulemeester (1 5) propõe um algoritmo exato que utiliza a estratégia de limitante inferior: inicialmente, aloca-se a menor quantidade de recursos possível para os diferentes tipos de recursos. A seguir, determina-se o mínimo *makespan* através do algoritmo *branch-and-bound* proposto por Demeulemeester (1 2) para o PPPRL. Se o valor do *makespan* é menor ou igual à data de entrega, então o programa é ótimo para o PCUR. Caso contrário, a quantidade de um dos recursos é aumentada de uma unidade e o procedimento é repetido. Testes computacionais indicam que este procedimento é superior ao proposto por Möhring (1 4). Rangaswamy (1) propõe um novo limitante inferior para o PCUR e utiliza o mesmo conjunto de problemas teste que Demeulemeester (1 5) para avaliar o desempenho de seu algoritmo. O procedimento de Rangaswamy (1) reduz a quantidade de PPPRL's que precisam ser resolvidos, em comparação com o procedimento de Demeulemeester (1 5). Entretanto, uma comparação dos tempos computacionais entre os dois métodos não é apresentada, pois os trabalhos utilizam diferentes plataformas computacionais e portanto, não é possível saber de fato a extensão da economia de tempo computacional. Drexl e Kimms (2001) propõem dois métodos para obter limitantes inferiores para o PCUR, uma relaxação lagrangeana e um procedimento baseado em geração de colunas. Testes computacionais foram realizados adaptando os problemas teste da biblioteca PSPLIB de problemas para o PPPRL da literatura.

Um exemplo do PCUR com 12 atividades (atividades 1 e 12 são atividades artificiais que indicam o início e o fim, respectivamente, do projeto) e m=3 tipos de recursos. A Figura 1 mostra as relações de precedência entre as atividades. A quantidade de recursos que cada atividade precisa para ser processada é descrita nas linhas 2, 3 e 4 da Tabela 1. A data de entrega do projeto é 20 (D=20).

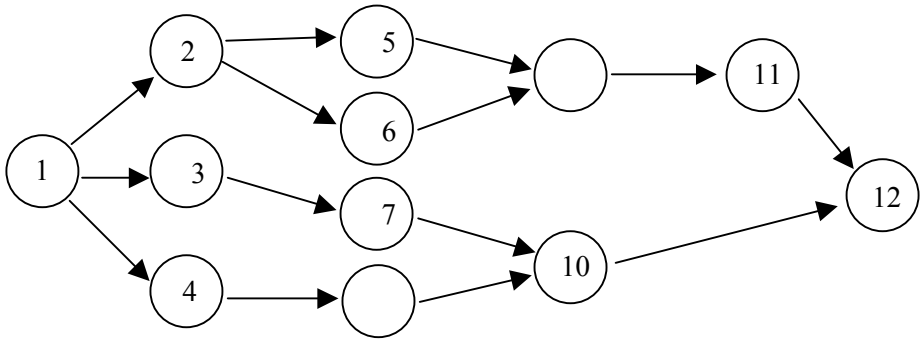

Figura 1. Relações de precedência de um projeto

Tabela 1. – Duração e recursos utilizados pelas atividades no exemplo.

Atividades	1	2	3	4	5	6	7			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	4	2	3	7	0

Suponha que sejam alocados a=(2,2,2) recursos. Um possível programa para esta solução pode ser visto na Figura 2. O eixo vertical corresponde aos tipos de recursos alocados e o eixo horizontal indica o instante de início e término de cada atividade. O projeto termina no instante 23, isto é, o projeto é infactível com respeito à data de entrega. Observe, entretanto, que o projeto é factível com respeito à restrição de recursos.

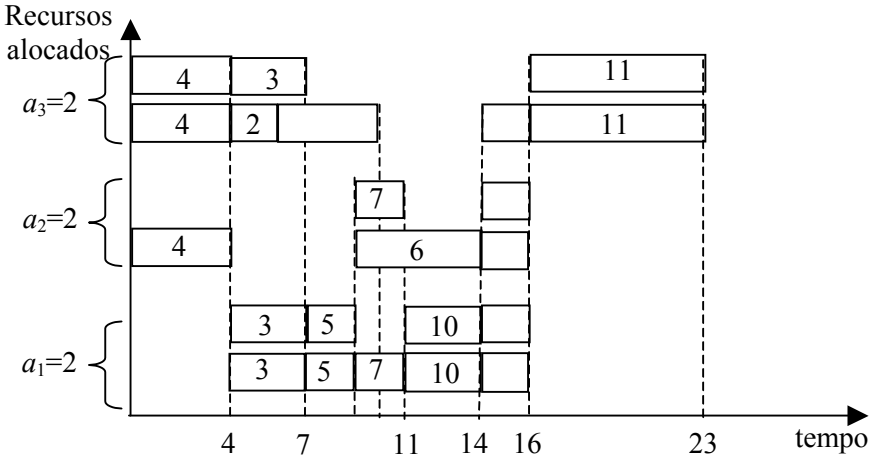

Figura 2. Programação das atividades

3. Avaliação da solução

A avaliação da solução é um importante passo para resolver o PCUR. Uma vez fixados os valores de recursos disponíveis, determina-se o programa (*schedule*) das atividades e calcula-se o instante de término do projeto. Este passo é equivalente à resolução de um PPPRL. Portanto, é necessário utilizar um procedimento de solução para o PPPRL que não seja muito custoso do ponto de vista computacional. As heurísticas do tipo *multi-pass,* baseadas em regras de prioridade, oferecem um bom compromisso entre qualidade de solução e desempenho computacional. Alguns estudos, como Kolisch e Drexl (1 6), propõe uma heurística *multi-pass* que utiliza amostragem aleatória. No método de amostragem aleatória, ao invés de calcular um valor de prioridade para cada atividade, calcula-se uma probabilidade $p(j)$, que corresponde à probablidade da atividade j ser selecionada, de tal forma que atividades que apresentem um maior valor de prioridade tenham maior probabilidade de

serem selecionadas. Neste trabalho, a heurística para resolver os PPPRLs segue a mesma linha da heurística de Kolisch e Drexl (1 6).

Uma vez obtido o programa de atividades, pode ser que exista folga na utilização dos recursos, isto é, os recursos alocados não são usados até o seu limite. Neste caso, realiza-se um ajuste para retirar as folgas, decrescendo a quantidade de recurso disponível até a folga desaparecer.

3.1 Heurística de melhoria para o PCUR

A heurística de melhoria é composta de duas fases: factibilização (Fase 1) e redução de recursos (Fase 2). A Fase 1 é utilizada quando a solução corrente é infactível em termos da data de entrega, isto é, existe atraso na conclusão do projeto. Nesta fase, faz-se uma tentativa de factibilizar a solução corrente, aumentando a quantidade de recursos disponíveis. A escolha dos recursos que devem ser aumentados é feita utilizando o tempo de início (S_i) e o tempo de início mais tarde (LS_i) das atividades da seguinte forma: Seleciona-se a atividade q com a maior diferença entre S_i e LS_i. Os recursos da solução corrente são então aumentados de r_{qk}, k=1,..,m, e a solução é avaliada. Se a solução ainda continua infactível, o processo é repetido utilizando, desta vez, os recursos da atividade com segundo maior valor de (S_i - LS_i). Se a solução continuar infactível, então a heurística pára. Caso contrário, se uma solução factível puder ser obtida por este processo, então prossegue-se para a Fase 2. A restrição no número de tentativas para factibilizar a solução tem o objetivo de limitar o tempo computacional gasto por esta heurística que é utilizada diversas vezes pelos métodos *scatter search* e *multi-start*.

A Fase 2 é aplicada quando a solução corrente é factível e consiste em reduzir a quantidade de recursos disponíveis, um por vez, de uma unidade. Se uma nova solução factível e de menor custo for encontrada, então a solução corrente é atualizada e a heurística prossegue, tentando reduzir os recursos. Se a redução de recursos não gerar mais melhoria, o algoritmo pára. O psedocódigo da heurística de melhoria é dado na Figura 3 e um exemplo da aplicação da heurística é apresentado a seguir.

Considere o projeto descrito no exemplo da seção 2. Para fins ilustrativos, suponha que a solução corrente seja igual a a=(2,2,2), e que o programa correspondente a esta alocação é infactível com respeito à data de entrega do projeto. A Tabela 2 mostra o tempo de início das atividades (linha S_i), tempo de início mais tarde (linha LS_i), e a última linha mostra a diferença entre o tempo de início e o tempo de início mais tarde. Seguindo a Fase 1 da heurística de melhoria, seleciona-se a atividade com maior valor de S_i-LS_i. Observando a última linha da Tabela 2, é possível notar que as atividades e 11 têm o maior valor de S_i-LS_i. Como existe empate, escolhemos uma delas aleatoriamente. Suponha que 11 seja escolhida. A atividade 11, como pode ser visto na Tabela 1, necessita da seguinte quantidade de recursos: $r_{11,1}$=0, $r_{11,2}$=0, $r_{11,3}$=2. Portanto, os recursos devem ser aumentados para a'=(2+0,2+0,2+2)=(2,2,4). Se não for possível encontrar uma solução factível para esta alocação, faz-se uma segunda tentativa, utilizando a atividade . A solução corrente a=(2,2,2) dá origem à solução a'=(2+2,2+2,2+1)=(4,4,3). Se as duas tentativas de factibilização não tiverem sucesso, a heurística pára, caso contrário, inicia-se a Fase 2.

Suponha que a alocação de recursos a = (5,3,3) dê origem a um programa factível com respeito à data de entrega do projeto. A Fase 2 da heurística de melhoria pode ser aplicada a esta solução:

Iteração 1: solução corrente: a = (5,3,3)

a'=(5-1,3,3) =(4,3,3) - solução infactível

a'=(5,3-1,3)=(5,2,3) - solução factível e de menor custo, logo a=(5,2,3).

Iteração 2: solução corrente: a = (5,2,3)

a' = (5-1,2,3) = (4,2,3) - solução infactível

a' = (5,2-1,3) = (5,1,3) - solução infactível
a' = (5,2,3-1) = (5,2,2) - solução infactível
FIM

Passo 1: Se solução é infactível, vá para o **Passo 2**, caso contrário vá para o **Passo 3**.
Passo 2: Factibilização da solução
Passo 2.1. Factibilização (primeira tentativa): Calcular S_i - LS_i e selecionar atividade p com máximo S_i - LS_i. Aumentar os recursos da solução corrente, $a = (a_1,...,a_m)$ acrescentando os recursos utilizados pela atividade p: $a'=(a_1 + r_{p1}, ..., a_m + r_{pm})$. Se a solução for factível, vá para o **Passo 3**. Se a solução for infactível, vá para o **Passo 2.2**.
Passo 2.2. Factibilização (segunda tentativa): Calcular S_i - LS_i e selecionar atividade q com segundo maior valor de S_i - LS_i. Aumentar os recursos da solução corrente, $a = (a_1,...,a_m)$ acrescentando os recursos utilizados pela atividade q: $a'=(a_1 + r_{q1}, ..., a_m + r_{qm})$. Se a solução for factível, vá para o **Passo 3**. Se a solução for infactível, encerrar algoritmo.
Passo 3: Redução de recurso.
Para cada recurso k=1,.., m, faça:
Reduzir o recurso k de uma unidade: $a'_k = a_k - 1$. Avaliar solução. Se a solução for factível e de menor custo que a solução corrente, atualizar a solução corrente $a_k = a'_k$.
Passo 4: Se houver melhoria no **Passo 3**, repetir **Passo 3**. Caso contrário, encerrar algoritmo.

Figura 3. Pseudocódigo da heurística de melhoria.

Tabela 2. – Exemplo da aplicação da Fase 1 da heurística de melhoria

Atividades	1	2	3	4	5	6	7			10	11	12
S_i	0	4	4	0	7			6	14	11	16	23
LS_i	4	4	12			6	15	13	11	17	13	20
S_i-LS_i	-	0	-	-	2	-3	-6	-7	3	-6	3	-

4. Heurística multi-start

A heurística *multi-start* é descrita a seguir.

Passo 1. Gerar os valores de recursos a_k, k=1,..., m.
Passo 2. Aplicar heurística de melhoria.
Passo 3. Repetir Passos 1 e 2 enquanto o número de soluções avaliadas for menor do que *MaxSol*, onde *MaxSol* é o número máximo de soluções avaliadas.

Foram testadas duas versões do método *multi-start*, *multi-start aleatório* (MSA) e *multi-start baseado em freqüência* (MSF). A diferença entre as duas versões reside no modo como o Passo 1 do algoritmo é realizado. No MSA, os valores a_k, k = 1,..., m, são gerados aleatoriamente por uma distribuição uniforme no intervalo (l_k , u_k) onde

$$l_k = \max_{i=1,...,n} \{r_{ik}\}, \qquad k = 1,..., m \qquad (6)$$

é um limitante inferior para o recurso k e

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

é um limitante superior.

A versão MSF usa informação das soluções encontradas com o objetivo de diversificar a busca. A idéia é armazenar os valores de a_k, k =1,..., m, que foram obtidos durante a busca e utilizar esta informação para evitar soluções repetidas. Para armazenar os valores de a_k, é necessário classificá-los em faixas, que pertencem a um intervalo de valores (x_0, x_g), dividido em g segmentos de tamanho igual:

Segmento 1: (x_0^k, x_1^k)
Segmento 2: (x_1^k, x_2^k)
Segmento 3: (x_2^k, x_3^k)
$\vdots$
Segmento g: (x_{g-1}^k, x_g^k)

Para todo valor de recurso a_k , é preciso encontrar a faixa de valores que corresponde a este recurso, isto é, encontrar o valor i tal que $x_i^k < a_k < x_{i+1}^k$. Feito isto, atualiza-se a matriz de freqüência $M[k][i]$, onde k=1,..., m corresponde ao tipo de recurso e i = 0,..,g corresponde à faixa a qual o recurso pertence. Os valores de recurso do Passo 1 do algoritmo acima são então gerados com probabilidade inversamente proporcional à freqüência de seus valores em M. Um esboço da heurística MSF é dado na Figura 4.

Passo 0. x_0^k = limitante inferior do recurso k, calculado por (6). Cálculo de x_g^k: gera-se aleatoriamente uma solução e aplica-se a heurística de melhoria, obtendo então o seguinte vetor de recursos $H=(h_1,...,h_m)$. O extremo x_g^k é dado então por: $x_g^k = \lambda \cdot h_k$, k = 1, ..., m, onde $\lambda \geq 1$ é um parâmetro. Inicializar $M[k][i]$=1, k=1,..., m e i = 0,.., g-1
Passo 1. Gerar os valores de recursos a_k, k=1,..., m.
Passo 2. Aplicar heurística de melhoria.
Passo 2.1. Atualizar a matriz M.
Passo 3. Repetir Passos 1 e 2, enquanto o número de soluções avaliadas for menor do que *MaxSol*

Figura 4. Heurística MSF.

5. 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 (1 77), e um modelo (*template*) detalhado do método foi apresentado em Glover (1). Métodos populacionais geram um conjunto de soluções a cada iteração através de operações específicas aplicadas ao conjunto de soluções da solução 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. O enfoque do método pode ser descrito abaixo de forma sucinta pelo seguinte modelo:

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; um subconjunto das melhores soluções obtidas é chamado de conjunto de referência.

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 do conjunto de referência; as soluções de cada subconjunto são combinadas.

5) Método para gerar combinação estruturada de soluções: uma combinação estruturada utiliza informação das soluções a serem combinadas de forma a gerar soluções dentro e fora da região convexa gerada pelos elementos do conjunto de referência; o método de melhoria é aplicado às melhores soluções geradas pelas combinações.

Os parâmetros utilizados no algoritmo SS são descritos a seguir:
PSize = tamanho do conjunto de soluções diversas gerados pelo método de geração de soluções diversas
b = tamanho do conjunto de referência
b_1 = tamanho do subconjunto de soluções de qualidade
b_2 = tamanho do subconjunto de soluções diversas
MaxIter = número máximo de iterações

5.1 Método de geração de soluções diversas

As soluções diversas são geradas utilizando a matriz de freqüência *M*, descrita na Seção 4. Os valores de recurso a_k, k=1,..., m são então gerados com probabilidade inversamente proporcional à freqüência de seus valores em *M*.

5.2 Método de criação e atualização do conjunto de referência

Este método é utilizado para criar e manter um conjunto de soluções de referência. Soluções são incluídas neste conjunto de referência de acordo com a sua qualidade ou diversidade. O subconjunto de qualidade corresponde às b_1 melhores soluções (em termos de valor de função objetivo) e o subconjunto diverso consiste de soluções dispersas.

Método de criação do conjunto de referência

Considere o *conjunto de soluções tentativas diversas* (CSTD) geradas por 1) no modelo. O subconjunto de soluções de qualidade é formado pelas b_1 melhores soluções pertencentes a CSTD. O subconjunto diverso é obtido através do seguinte procedimento: calcula-se a distância mínima de todos os elementos de CSTD, que no momento não pertençam ao conjunto de referência, e seleciona-se o elemento que maximiza a distância mínima a todas as soluções deste conjunto. O procedimento deve ser repetido até que o subconjunto diverso tenha b_2 elementos. Neste estudo, a distância entre duas soluções, x e y, é medida por,

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

Método de atualização do conjunto de referência

Considere uma solução gerada por 5) do modelo. Se o valor da solução for menor que o valor de um dos elementos do subconjunto de qualidade, então esta solução substitui o pior elemento do subconjunto. Caso contrário, esta solução entra no subconjunto de soluções diversas se aumentar a menor distância entre as soluções deste subconjunto.

5.3 Geração de subconjuntos

Neste trabalho, utilizou-se o mesmo método de geração de subconjuntos descrito em Glover (1). Quatro tipos de subconjuntos podem ser gerados:

1) Subconjuntos formados pela combinação das soluções de referência duas a duas.
2) Subconjuntos de 3 elementos que são obtidos a partir dos subconjuntos de dois elementos adicionando a melhor solução (em termos de valor de função objetivo) que não se encontra neste subconjunto.
3) Subconjuntos de 4 elementos que são obtidos a partir dos subconjuntos de três elementos adicionando a melhor solução (em termos de valor de função objetivo) que não se encontra neste subconjunto.
4) Subconjuntos que consistem dos i melhores elementos (medidos através do valor de função objetivo), para i=5 até b.

5.4 Método de combinação de solução

Este procedimento transforma um dado subconjunto de soluções produzido pelo método de geração de subconjuntos em uma ou mais soluções combinadas. Nesta versão do algoritmo SS foi

utilizado um método de combinação que cria somente uma solução a partir de cada subconjunto. Sejam $a_1^j,....,a_m^j,\ j=1,...,r$ as soluções no subconjunto *P*. As soluções combinadas são dadas por,

$$a_k = \left\lfloor \frac{\sum_{j\in P}(1/VO(j))\cdot a_k^j}{\sum_{j\in P}(1/VO(j))} \right\rfloor \qquad k=1,...,m$$

onde *VO*(*j*) é o valor da função objetivo da solução *j* e $\lfloor x \rfloor$ denota o maior inteiro incluído em *x*. A idéia deste tipo de combinação é dar um peso maior para as soluções que apresentem um melhor valor de função objetivo.

6. Resultados computacionais

Esta seção descreve como os problemas teste foram gerados e avalia o desempenho computacional das heurísticas SS, MSA e MSF. Os experimentos computacionais foram realizados num PC Pentium III, 667MHz, 256 Mbyte RAM. Todos os procedimentos foram programados na linguagem C++.

6.1 Geração de problemas para o PCUR

Em geral, os problemas teste para o PCUR são gerados através de adaptações de problemas teste para o PPPRL. Isso ocorre porque os problemas PCUR e PPPRL têm vários dados de entrada em comum, possibilitando uma adaptação do método de geração de PPPRL já existente, para o problema aqui tratado. O programa Progen (Kolisch *et al.*, 1 5) é um gerador de problemas teste para o PPPRL bastante conhecido e foi utilizado para gerar os problemas teste da biblioteca PSPLIB, que contém problemas com 4 recursos e 30, 60, 0 e 120 atividades. Neste trabalho, testes computacionais são realizados com problemas teste de 30, 60, 0 e 120 atividades e 4, 6 e recursos, geradas pelo programa Progen e adaptadas para o PCUR seguindo a metodologia de Drexl e Kimms (2001). Os problemas teste com 4 recursos foram retirados diretamente da biblioteca PSPLIB e os demais foram geradas pelo Progen. Os limites dos recursos não têm qualquer significado para nossas aplicações e portanto foram ignorados. Dois parâmetros importantes do Progen são a complexidade da rede (*network complexity* - *NC*) e o fator de recurso (*resource factor* - *RF*). A complexidade da rede 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 utilizados por uma atividade. Por exemplo, para *RF* = 1, cada atividade requer todos os *m* tipos de recursos enquanto para *RF* = 0, a atividade não requer qualquer tipo de recurso. Assim como nos trabalhos que abordam o PPPRL e nos problemas teste da biblioteca PSPLIB, são utilizados aqui quatro valores de *RF*: 0.25, 0.5, 0.75 e 1.0 e três valores de *NC*: 1.5, 1. e 2.1. A combinação de *RF*, *NC*, *m* e *n* resulta num total de 4*3*3*4 = 144 problemas teste. É necessário ainda determinar um limite de tempo para o projeto. Drexl e Kimms (2001) sugerem que a data de entrega *D* para o projeto seja calculada utilizando o caminho crítico da rede,

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

onde $DF \in \{1.0,1.2,1.4\}$ é o **fator de data de entrega**, e EF_i é o instante de término mais cedo da atividade *i*. Isso dá origem a um banco de dados com 144*3 = 432 problemas teste. Para cada problema teste foram gerados custos C_k com distribuição uniforme no intervalo [1,10].

O valor dos parâmetros utilizados pela heurística SS foram: ***PSize*** = 30, $\boldsymbol{b}$ = 5, $\boldsymbol{b_1}$ = 3, $\boldsymbol{b_2}$ = 2 e ***MaxIter***=1, isto é, SS pára após realizar uma iteração completa. Este limite de *MaxIter*=1 deve-se ao tempo computacional requerido para resolver problemas grandes. Para MSA e MSF, utilizou-se o seguinte valor para os parâmetros: $\boldsymbol{g}$=10 e $\boldsymbol{\lambda}$=1. 0. O critério de parada adotado para MSA e MFS (*MaxSol*) foi baseado no número de soluções avaliadas por SS. A Tabela 3 mostra o número de

soluções avaliadas durante a execução de SS, para uma determinada quantidade de atividades, recursos e fator de data de entrega.

Tabela 3. Número máximo de soluções avaliadas

	***m* = 4**			***m* = 6**			***m* = 8**		
n	***DF*=1.0**	***DF*=1.2**	***DF*=1.4**	***DF*=1.0**	***DF*=1.2**	***DF*=1.4**	***DF*=1.0**	***DF*=1.2**	***DF*=1.4**
30	4.16E+05	6.53E+05	5.7 E+05	7.62E+05	7.22E+05	7.57E+05	7.61E+05	. E+05	.6 E+05
60	.4 E+05	.03E+05	7.06E+05	.25E+05	1.07E+06	. 5E+05	1.01E+06	1.25E+06	1.07E+06
90	.0 E+05	1.16E+06	1.17E+06	1.43E+06	1.54E+06	1.56E+06	1.5 E+06	1.5 E+06	1.77E+06
120	1.45E+06	1.46E+06	1.30E+06	1.61E+06	1. 6E+06	1. 7E+06	2.12E+06	2.16E+06	2.0 E+06

6.2 Resultados computacionais para o PCUR

Esta seção discute o desempenho das heurísticas SS, MSA e MSF para os problemas teste propostos. Para um dado problema teste, seja *Best* o custo total (veja equação 1) da melhor solução, dentre as soluções obtidas por SS, MSA e MSF. A avaliação do desempenho das heurísticas SS, MSA e MSF foi realizada utilizando as seguintes medidas:

- Desvio relativo: (*H-Best*)/(*Best*), onde *H* é o custo total da solução encontrada pela heurística.
- Desvio médio: média dos desvios relativos (em %)
- Desvio máximo: maior valor de desvio relativo de *H* com relação a *Best*. (em %)
- Número de vitórias: número de problemas teste onde *H* é igual a *Best*.

A Tabela 4 apresenta um resumo do desempenho de SS, MSF e MSA, de acordo com o número de atividades. Dentre as três heurísticas, MSA foi a que obteve os piores resultados. Comparando o desempenho de MSA com MSF, é interessante notar que MSF apresenta resultados, consistentemente, melhores do que MSA, mostrando a importância da utilização da estrutura de memória (matriz de freqüência). Observando a Tabela 4 é interessante notar que dentre as três heurísticas, a heurística SS foi a que apresentou o melhor desempenho para 60, 0 e 120 atividades. Nos problemas teste com 30 atividades, entretanto, MSF parece ter um desempenho um pouco melhor do que SS. Resultados computacionais mais detalhados para os problemas teste de 30 atividades são ilustrados na Tabela 5. Esta tabela mostra o desempenho de SS, MSF e MSA, discriminando os problemas teste por número de recursos, 4, 6 e , e fatores de data de entrega, 1.0, 1.2 e 1.4.

Tabela 4. Resumo do desempenho das heurísticas

	SS			**MSF**			**MSA**			**Núm. de problemas teste**
n	**Num. Vitórias**	**Desvio médio (%)**	**Maior desvio (%)**	**Num. Vitórias**	**Desvio Médio (%)**	**Maior desvio (%)**	**Num. Vitórias**	**Desvio médio (%)**	**Maior desvio (%)**	
30	6	0.42	4.15	74	0.45	3.66	51	1.03	.17	10
60	77	0.27	3.02	3	1.02	7.62	27	1. 3	13.61	10
90	5	0.20	3.34	2	1.62	.11	13	3.1	1 .00	10
120	5	0.0	1.73	20	2.05	10.56	13	4.30	1 .14	10

Note que para os problemas teste com 6 recursos, na Tabela 5, a heurística MSF apresenta um número de vitórias maior do que SS para todos os valores de DF.

Um teste interessante para avaliar o desempenho de SS é verificar a influência do método de geração do conjunto diverso e método de combinação (tipo 1, 2, 3 e 4) implementados. O objetivo do gráfico da Figura 5 é mostrar o número de casos nos quais a melhor solução de um problema teste foi encontrada pelo SS durante a execução do método de geração do conjunto diverso, combinação do tipo 1, combinação do tipo 2, combinação do tipo 3 ou combinação do tipo 4. Observando o gráfico

pode-se notar que para os problemas teste com 30 e 60 atividades, a melhor solução é encontrada, freqüentemente, na fase de geração do conjunto diverso. Entretanto, para problemas teste de maior porte, como 0 e 120 atividades, as soluções parecem se concentrar nas combinações do tipo 1 e 3. É interessante notar também que, aparentemente, o número de soluções encontradas durante a fase de geração do conjunto diverso descresce à medida que o número de atividades aumenta, em contraste com o que ocorre durante a fase de combinação do tipo 3.

Tabela 5. Problemas teste com 30 atividades

		SS			MSF			MSA		
	DF	**Num. Vitórias**	**Desvio médio (%)**	**Desvio máx. (%)**	**Num. Vitórias**	**Desvio médio (%)**	**Desvio máx. (%)**	**Num. Vitórias**	**Desvio médio (%)**	**Desvio máx. (%)**
	1.00	10	0.23	1.61	10	0.15	1.20		0.50	4.1
***M*=4**	**1.20**		0. 0	4.15		0.45	2.41	5	1. 4	.17
	1.40		0.25	2. 1	6	0. 7	3.66	7	1.37	7. 2
	1.00		0.41	1.7	11	0.17	2.00	5	1.0	5.20
***M*=6**	**1.20**	5	0.6	2. 5	7	0.72	3.55	4	0.	3.0
	1.40	7	0.33	1. 0	10	0.30	2. 6	5	0. 2	3.30
	1.00	7	0.2	1.56	7	0.44	1. 7	6	0.72	2.03
***M*=8**	**1.20**	6	0.44	1.51	7	0.42	3.07	6	0.65	2.74
	1.40		0.35	2.5		0.42	1. 3	4	1.43	4.73
média			0.42			0.45			1.03	
soma		6			74			51		

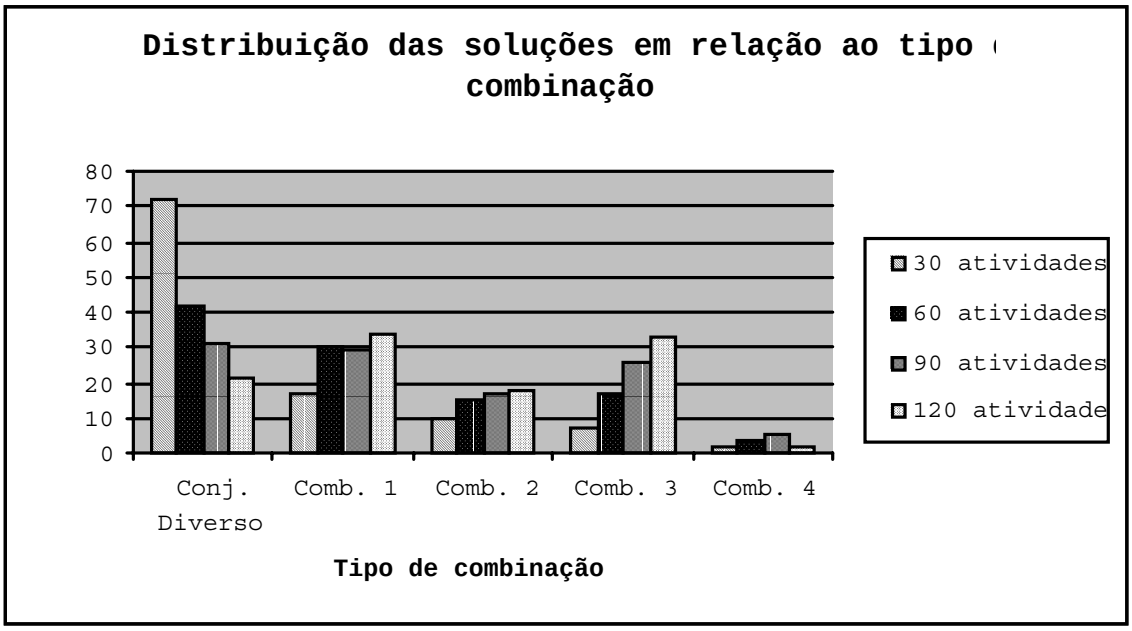

Figura 5. Histograma das soluções do SS em termos de conjunto diverso e tipo de combinação

O tempo computacional gasto pelas heurísticas, em cada problema teste, foi bastante semelhante. A Tabela 6, descreve o tempo computacional médio gasto por uma heurística para resolver um problema teste de acordo com o número de atividades, recursos e fator de data de entrega.

Observe que os tempos computacionais, em geral, crescem à medida que o número de atividades, fator de data de entrega e número de recursos aumentam.

Tabela 6. Média dos tempos computacionais (em segundos)

n	*m*								
	***DF*=1.0**			***DF*=1.2**			***DF*=1.4**		
	4	**6**	**8**	**4**	**6**	**8**	**4**	**6**	**8**
30	16. 5	25.63	23.25	37.10	33.46	36.35	41. 1	45.70	46.27
60	1. 3	5.72	75.26	7.75	125.65	11 .77	117. 1	161.26	144.17
90	161.5	231.0	23 .	277.04	333.5	347.5	330. 2	3 0.1	443.54
120	471. 4	567.30	476.46	513.56	675.51	66 .34	66 .16	7 5.2	23.4

7. Conclusão

O objetivo deste trabalho foi abordar o problema de programação de projetos conhecido na literatura como PCUR. Uma heurística, baseada no método *scatter search* (SS), e duas versões do algoritmo *multi-start* (MSA e MSF) são propostas. Com relação ao desempenho das heurísiticas *multi-start*, observou-se nos testes computacionais que MSF apresentou resultados melhores do que MSA. A diferença básica entre MSA e MSF foi a utilização de uma estrutura de memória em MSF, que teve como finalidade obter uma maior diversificação durante o procedimento de busca. Esta estrutura é responsável pela vantagem de MSF em relação a MSA. A heurística SS apresentou, em geral, um desempenho melhor do que MSA e MSF, principalmente para problemas de médio e grande porte. No futuro, seria interessante estudar o efeito de outros métodos de combinação para o SS, visto que as combinações demonstraram ser um fator importante na qualidade de soluções de problemas teste de médio e grande porte.

Agradecimento

Esta pesquisa teve o apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

Blazewics, J., Lenstra, J.K., e A.H.G. Rinnooy Kan (1983). "Scheduling subject to resource constrains: classification and complexity," *Discrete Applied Mathematics*, 5, 1 3, 11-24.

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

Demeulemeester, E. (1995). "Minimizing resource availability costs in time-limited project networks," *Management Science*, Vol. 41, No. 10, 15 0-15 .

Demeulemeester, E., e S. Herroelen (1992). "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem", *Management Science*, Vol. 3 , No. 12, pp. 1 03-1 1 .

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

Glover, F. (1998) "A Template for Scatter Search and Path Relinking," *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), 13-54, Springer.

Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints,"*Decision Sciences*, , 156-166.

Herroelen, W., De Reyck B., e E. Demeulemeester (1998). "Resource-constrained project scheduling: A survey of recent developments," *Computers Ops. Res.*, Vol. 25, No. 4, 27 -302.

Kolisch, R., e A. Drexl (1996). "Adaptative search for solving hard project scheduling problems," *Naval Research Logistics*, 43, 23-40.

Kolisch, R., Sprecher, A., e A. Drexl (1995). "Characterization and generation of a general class of resource-constrained project scheduling problems," *Management Science*, Vol. 41, No.10, 16 3-1703.
Kolisch, R., e S. Hartmann (1998). "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis," *Project Scheduling: recent models, algorithms,and applications*, 147-17 , Jan Weglarz (ed.). Kluwer Academic Publishers.
Moder, J.J., Phillips, C.R. e E.W. Davis (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, -120.
Neumann, K., Schwindt, C. e J. Zimmermann. "Project scheduling with time windows and scarce resources: Temporal and resource-constrained project scheduling with regular and nonregular objective functions", Springer-Verlag Berlin heidelBerg, 2002.
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*.