XLV SBPO
Simpósio Brasileiro de Pesquisa Operacional
Natal - RN
16 a 19 de setembro de 2013

Minicurso
Modelos e Métodos de Resolução para Problemas de Escalonamento de Projetos

Haroldo Gambini Santos

Departamento de Computação
Universidade Federal de Ouro Preto - Campus Universitário
haroldo@iceb.ufop.br

Túlio Ângelo Machado Toffolo

Departamento de Computação
Universidade Federal de Ouro Preto - Campus Universitário
tulio@toffolo.com.br

Marco Antonio Moreira de Carvalho

Departamento de Computação
Universidade Federal de Ouro Preto - Campus Universitário
marco.opt@gmail.com

Janniele Aparecida Soares

Departamento de Computação
Universidade Federal de Ouro Preto - Campus Universitário
jannysdp@hotmail.com

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Siglas

AG	Algoritmos Genéticos
AoA	*Activity-on-arc*
AoN	*Activity-on-node*
CPM	*Critical Path Method*
EFT	*Earliest Finish Time*
EFTD	*Dynamic Earliest Finish Time*
EST	*Earliest Start Time*
ESTD	*Dynamic Earliest Start Time*
FBI	*Forward-Backward Improvement*
FF	*Free Float*
GOAL	Grupo de Otimização e Algoritmos
LST	*Latest Start Time*
LFT	*Latest Finish Time*
MISTA	*Multidisciplinary International Scheduling Conference*
MSLK	*Minimum Slack*
MSLKD	*Dynamic Minimum Slack*
MMRCPSP	*Multi-Mode Resource-Constrained Project Scheduling Problem*
MRCMPSP	*Multi-Mode Resource-Constrained Multi-Project Scheduling Problem*
PERT	*Program Evaluation and Review Technique*
PI	Programação Inteira
PSP	*Project Scheduling Problem*
PSPLIB	*Project Scheduling Problem Library*
RCPSP	*Resource-Constrained Project Scheduling Problem*
RSM	*Resource scheduling method*
SBSE	*Search-Based Software Engineering*
SGS	*Schedule Generation Scheme*
SMRCPSP	*Single-Mode Resource-Constrained Project Scheduling Problem*
TF	*Total Float*
TPD	*Total Project Delay*
TMS	*Total Makespan*

1 Introdução

No contexto atual da sociedade e no ambiente competitivo, é fundamental oferecer produtos e serviços de alta qualidade levando em consideração tempo e orçamento viáveis. Garantir a boa utilização de recursos e o cumprimento de prazos requer um bom planejamento para realizar as atividades que originam os produtos e serviços.

Primeiramente deve-se entender que um projeto é composto por um conjunto de atividades, onde cada atividade possui relações de precedências e requer tempo e recursos para ser executada. O Problema de Escalonamento de Projetos, ou PSP (*Project Scheduling Problem*), consiste em programar as atividades de forma que nenhuma infrinja suas relações de precedências e não extrapole as quantidades de recursos disponíveis. Cada atividade pode possuir um ou mais modos de execução, com diferentes combinações de duração e consumo de recursos.

Algumas variações do PSP consideram ainda algumas questões além da dependência entre atividades e múltiplos modos de execução de cada atividade, como a utilização de recursos renováveis e não-renováveis e o compartilhamento destes recursos entre diferentes projetos. Neste minicurso, além da apresentação de conceitos fundamentais, apresentaremos uma versão abrangente do problema, conhecida como Escalonamento de Múltiplos Projetos com Múltiplos Modos e Recursos Limitados.

O objetivo, em geral, é encontrar uma solução factível que minimize o tempo de execução dos projetos envolvidos nas instâncias.

O problema é da classe NP-Difícil demonstrado em [Demeulemeester (2002)]. É utilizado para modelar inúmeros problemas em diversas áreas, tais como gestão de projetos em empresas de tecnologia da informação, agendamento de instruções para arquitetura de processadores, construção civil, programação de produção de lingotes de rolamento, montagem de loja, entre outros. Ademais, o progresso na resoluções de problemas NP-Difíceis é meta de inúmeras pesquisas em computação, matemática e pesquisa operacional, o que contribui para o aumento da motivação deste trabalho.

Um modelo genérico para descrever um PSP composto por diversos projetos e restrições foi proposto pela MISTA2013 (*Multidisciplinary International Scheduling Conference, 2013*) tornando-se desafiador para a comunidade acadêmica.

Métodos baseados em programação inteira já foram propostos para o PSP, no entanto, esses métodos são capazes de resolver apenas um subconjunto bastante restrito de instâncias em um tempo viável. Tendo em vista estas considerações, o conteúdo deste minicurso inclui, além de formulações, métodos heurísticos. São apresentadas heurísticas construtivas gulosas para a geração de uma solução inicial e heurísticas de busca local para o Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Recursos Limitados.

As técnicas apresentadas neste minicurso consideram instâncias específicas dos projetos que estão disponível no site da PSPLIB[1] (*Project Scheduling Problem Library*) e instâncias dos múltiplos projetos disponíveis no site da competição MISTA2013[2]. Assim qualquer instância especificada neste formato pode ser trabalhada pelas técnicas apresentadas.

[1] http://www.om-db.wi.tum.de/psplib/, acessado em Julho de 2013.

[2] http://allserv.kahosl.be/mista2013challenge/, acessado em Julho de 2013.

2 Representações de Projetos em Redes

Dois esquemas de representação de projetos em redes são usados para o PSP: representação por atividades em arcos e representação por atividades em nós. Esses esquemas propõem maneiras diferentes para representar as atividades e os eventos que compõem os projetos. Os eventos definem relações existentes entre as atividades. As subseções seguintes detalham ambas as representações.

2.1 Atividades em arcos - AoA

A representação de projetos por atividades em arcos é dada por um grafo direcionado $G = (N, A)$, onde os nós $n \in N$ representam os eventos e os arcos $a \in A$ representam as atividades.

Um projeto representado por AoA (*Activity-on-arc*) possui as seguintes caracteristicas:

- uma atividade só pode começar a ser executada após todas as atividades que a precedem;
- os arcos implicam as precedências lógicas existentes na rede;
- os nós devem ser enumerados de forma única;
- uma atividade é definida por seu início e fim através dos nós, sendo que dois nós podem ser ligados diretamente apenas por uma única atividade;
- a rede AoA não pode conter ciclos;
- cada rede AoA deve possuir apenas um nó inicial conectado aos nós que não possuem predecessores e apenas um nó final conectado aos nós que não possuem sucessores.

Atividades e eventos fictícios podem ser criados para contribuir com três casos:

- preservar exclusividade das atividades;
- preservar a condição de evento inicial e final único;
- representar corretamente todas as relações de precedências.

Na figura 1 é apresentado um exemplo de um projeto em rede AoA.

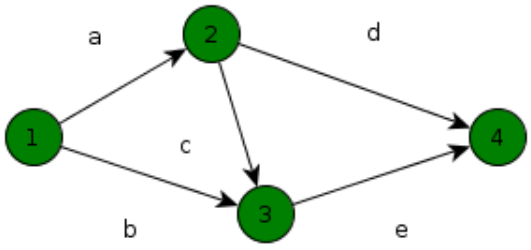

Figura 1. Representação Rede de Projeto AoA

2.2 Atividades em Nós - AoN

A representação de projetos por atividades em nós é dada por um grafo direcionado $G = (N, A)$, onde cada atividade é representada pelos nós $n \in N$ e cada arco direcionado $a \in A$ representa uma relação de precedência entre duas atividades.

Um projeto representado por AoN (*Activity-on-node*) possui três vantagens apresentadas em [Slack (2013)]:

- não necessita de atividades fictícias para manter a lógica de relacionamentos;
- a maioria dos pacotes computacionais usados para o planejamento de projetos utilizam a representação AoN;
- é, normalmente, mais fácil transformar a lógica básica de relacionamento de projeto em representação AoN do que em representação AoA.

Na figura 2 é feita a representação AoN da figura 1.

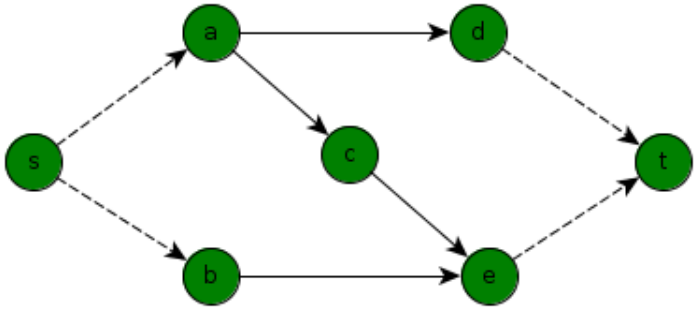

Figura 2. Representação Rede de Projeto AoN

3 Análise Temporal

Nesta seção são apresentados dois métodos baseados em rede que auxiliam no planejamento, escalonamento e controle de projetos. Lembrando que um projeto é definido como um conjunto de atividades inter-relacionadas, sendo que cada atividade consome tempo e recurso.

Os métodos a serem estudados são CPM (*Critical Path Method*) e PERT (*Program Evaluation and Review Technique*). Ambos fornecem meios analíticos para programar as atividades, porém o CPM considera durações determinísticas das atividades, enquanto o PERT considera durações probabilísticas.

As etapas que compõe os métodos são:

- definir as atividades do projeto;
- definir as relações de precedência;
- definir os requisitos de tempo;
- representar através de uma rede as relações de precedências;
- fazer cálculos específicos para desenvolver uma programação temporal;
- fazer revisões entre a fase de rede e programação temporal.

3.1 O Método do Caminho Crítico

O caminho crítico é definido por uma sequência de atividades críticas que partem do início ao fim do projeto. Uma atividade é denominada crítica caso não possua folga, que é o tempo disponível de deslocamento dentro de um intervalo. Estas atividades precisam ser concluídas dentro do instante de tempo esperado, pois o seu deslocamento em relação ao início e fim interfere diretamente na conclusão do projeto como um todo. Uma atividade não crítica possui folga, o que permite o seu deslocamento para frente ou para trás nos instantes de tempo sem interferir na conclusão do projeto.

O caminho crítico é um dos principais gargalos em escalonamento de projetos, visto que ele reflete diretamente no tempo de conclusão do mesmo. O resultado final do CPM é a construção da programação temporal para o projeto, o método não leva em consideração as restrições de recursos.

Para obter o caminho crítico é necessário calcular as variáveis EST (*Earliest Start Time*), LST (*Latest Start Time*), EFT (*Earliest Finish Time*) e LFT (*Latest Finish Time*), que representam respectivamente, o tempo mais cedo e o mais tarde em que uma atividade pode iniciar e o tempo mais cedo e o mais tarde em que uma atividade pode terminar. Uma atividade j é denominada crítica, caso o $EST_j = LST_j$ e o $EFT_j = LFT_j$.

Dada uma rede $G = (N, A)$, N representa o conjunto de atividades j e A representa as relações entre as atividades j, Pj conjunto de atividades antecessoras e Sj um conjunto de atividades sucessoras imediatas a j, dj duração da atividade j e T limite superior permitido para o tempo de conclusão do projeto. Os passos para cálculo do caminho crítico consistem em obter os valores para as variáveis apresentadas anteriormente através do pseudocódigo apresentado no Algoritmo 1.

Algoritmo 1: EST, EFT, LST, LFT com passos *forward* e *backword*

Entrada: Predecessores P_j, Sucessores S_j, Duração d_j, Parâmetros n
Saída: EST, EFT, LST, LFT

```
1  EST_1 = EFT_1 = 0;
2  j = 2;
3  enquanto j ≤ n faça
4      EST_j ← max(EFT_i ∀i ∈ P_j);
5      EFT_j ← EST_j + d_j;
6      j++;
7  LST_n = LFT_n = T;
8  j = n − 1;
9  enquanto j ≥ 1 faça
10     LFT_j ← min(LST_i ∀i ∈ S_j);
11     LST_j ← LFT_j − d_j;
12     j−−;
```

Depois de obter os resultados, é possível determinar uma duração mínima para o projeto. Basta somar as durações das atividades que fazem parte do maior caminho desde o início até o fim do projeto. Na figura 3 é apresentada uma rede AoN e seu caminho crítico. Cada atraso causado a uma atividade crítica irá refletir em um atraso no projeto global. Do mesmo modo, uma redução no tempo do projeto global, só pode ocorrer caso a redução seja feita em uma atividade crítica.

3.2 Construção da Programação Temporal

Através dos cálculos obtidos na seção anterior, pode-se reconhecer que para uma atividade j o EST_j representa o tempo mais cedo de início e LFT_j representa o tempo mais tarde de conclusão. Isso significa que (EST_j, LFT_j) determina o intervalo máximo durante o qual a atividade poderá ser programada sem atrasar o projeto inteiro. Duas observações são necessárias para obter uma programação temporal [Demeulemeester (2002)]:

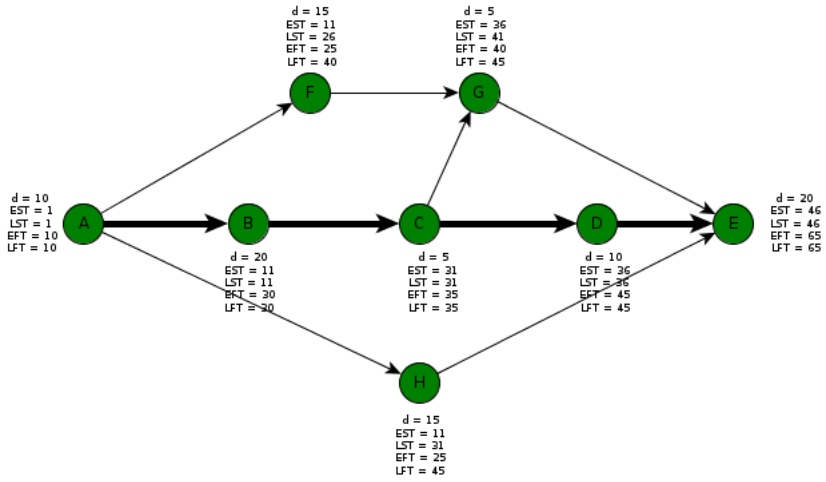

Figura 3. Caminho crítico

- as atividades críticas devem ser alinhadas uma depois da outra para garantir que o projeto seja concluído dentro de sua duração especificada (as atividades críticas são representadas por linhas em negrito);
- as atividades não críticas possuem intervalos de tempo maiores do que suas respectivas durações, o que permite uma folga para programá-las dentro de seus intervalos de tempo permitidos (as atividades não críticas são representadas pelas linhas que não estão em negrito).

Normalmente as atividades não críticas são iniciadas o mais cedo possível, caso todas as atividades possam ser programadas o mais cedo possível, a programação resultante é automaticamente viável. Caso não seja, algumas relações de precedência serão violadas e as atividades não críticas deverão ser adiadas para além de seu tempo mais cedo de início.

Duas folgas conhecidas são apresentadas, folga total (TF_j) e folga livre (FF_j), tal que $FF_j <= TF_j$. São elas:

- folga total (TF_j): define o excesso do intervalo de tempo definido entre a ocorrência mais cedo e mais tarde da atividade i;

$$TF_j = LST_j - EST_j = LFT_j - EFT_j$$

- folga Livre (FF_{ij}): define o atraso permitido para o fim da atividade j, tal que não afete o tempo de início de suas atividade sucessoras imediatas.

$$FF_j = min(EST_i) - EFT_j \forall i \in S_j$$

Para uma atividade não crítica j temos a **regra da bandeira vermelha**, que define:

- se $FF_j = TF_j$ então a atividade pode ser programada em qualquer tempo dentro de seu intervalo (EST_j, LFT_j) sem causar conflito na programação;
- se $FF_j \leq TF_j$, então a atividade pode ser atrasada por no máximo FF_j em relação ao seu tempo mais cedo de início, sem causar conflito de programação — qualquer

atraso maior que FF_j e menor que TF_j deve ser acoplado com um atraso igual em relação ao EST_j no tempo de início de todas as atividades que se original no nó j.

A figura 4 apresenta a programação temporal e as atividades críticas para o projeto de rede da figura 3.

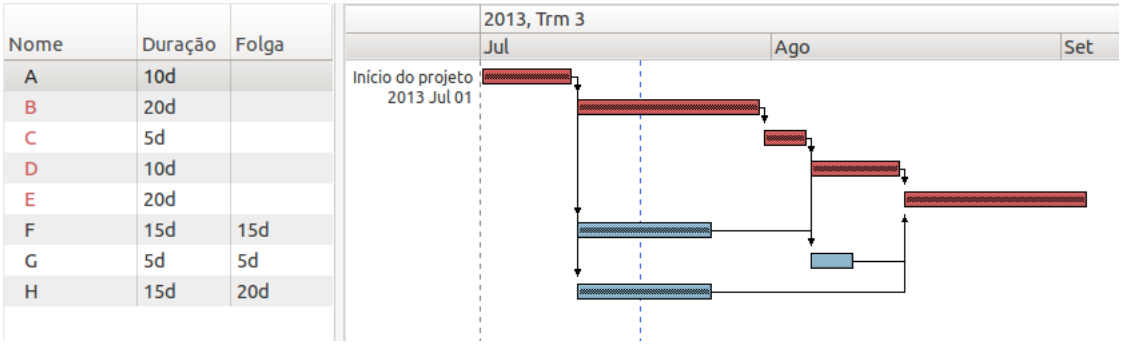

Figura 4. Programação Temporal

3.3 Técnica de Revisão e Avaliação de Programa

PERT (*Program evaluation and review technique*) é uma técnica de revisão e avaliação de programa que fornece meios analíticos para programar as atividades e é comumente utilizada em conjunto com o CPM. Esta técnica diferencia-se do CPM, pois baseia-se em três estimativas sobre a duração de uma atividade [Demeulemeester (2002)]:

- tempo otimista a: é o tempo mínimo possível para se realizar uma atividade;
- tempo mais provável m: é a melhor estimativa do tempo necessário para realizar uma atividade, em um ambiente normal;
- tempo pessimista b: é o tempo máximo possível para se realizar uma atividade.

A faixa (a, b) engloba todas as estimativas possíveis da duração de uma atividade, já a estimativa m encontra-se em algum ponto nesta faixa. É possível obter o tempo médio de duração $\tilde{D}$, que é uma estimativa sobre o tempo necessário para realizar uma atividade e a variância v através das fórmulas:

$$\tilde{D} = (a + 4m + b)/6$$

$$v = ((b - a)/6)^2$$

Tendo esses valores é possível estimar a probabilidade de que um nó j da rede ocorra conforme uma programação temporal pré-especificada, S_j.

A vantagem do PERT é que ele define explicitamente as relações de precedência, facilitando a identificação do caminho crítico e a identificação de um início precoce, início tardio e folga para cada atividade, bem como prevê uma duração potencialmente reduzida para o projeto. Um grande problema é que a rede do projeto pode não ser fácil de visualizar.

A probabilidade é dada por uma variável aleatória normal padronizada z tal que ela seja menor ou igual à K_j, definida pela fórmula:

$$K_j = (S_j - E(e_j))/\sqrt{var(e_j)}$$

Onde e_j é o tempo de ocorrência mais cedo do nó j, $E(e_j)$ representa a média e $var(e_j)$ representa a variância.

Os cálculos do CPM podem ser aplicados diretamente, visto que $\tilde{D}$ pode ser substituido por D.

4 Definição do Problema

De forma abrangente, pode-se dizer que para realizar o escalonamento do projeto, cada atividade deve ser alocada em um determinado instante de tempo e modo. O modelo considerado na PSPLIB permite que uma grande quantidade de problemas reais sejam fielmente modelados. Os diversos problemas possuem um determinado número de atividades, um horizonte de tempo, bem como a quantidade de modos que cada atividade pode ser executada e relações de precedência bem definidas, nas instâncias ainda são disponibilizadas as informações detalhadas sobre recursos disponíveis.

Na figura 5 é apresentado um grafo gerado a partir de uma instância de múltiplos projetos a serem escalonados simultaneamente.

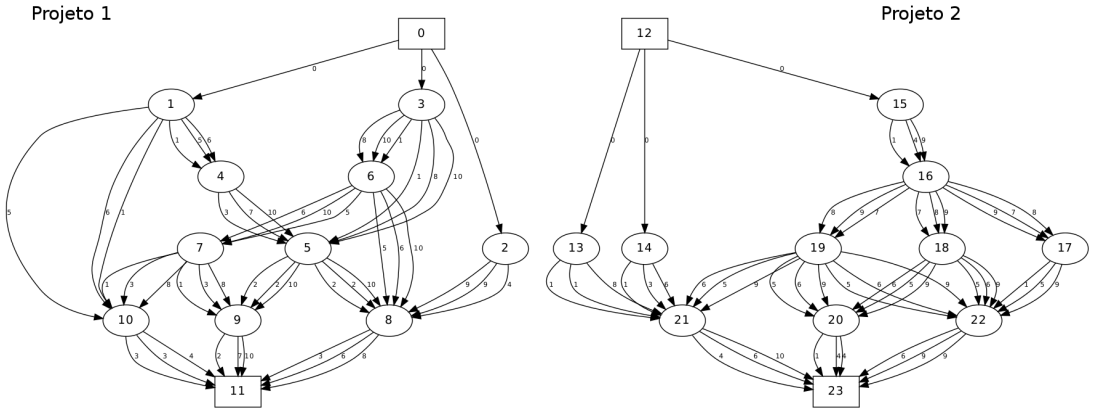

Figura 5. Representação por Grafo Instância A-1

A figura 6 apresenta uma solução ótima do escalonamento dos projetos da figura 5.

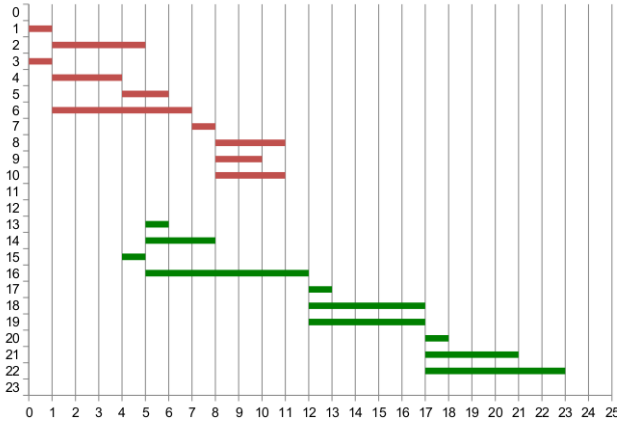

Figura 6. Representação Gráfica da Solução Instância A-1

4.1 Tipos de Problemas

Alguns tipos de problemas são apresentados na literatura — na PSPLIB existem instâncias disponíveis para cada tipo de problema. [Kolisch (1996)] define dois tipos de PSP. São eles:

- *single-mode* (SMRCPSP): cada uma das atividades do projeto tem de ser realizada de uma forma prescrita usando os recursos fornecidos — as atividades pertencentes ao projeto possuem apenas um modo;
- *multi-mode* (MMRCPSP): permite que as atividades sejam executadas em vários modos — estes modos possuem diferentes alternativas de combinações de duração e quantidade de recursos necessários para a execução da atividade.

Um novo problema derivado do *multi-mode* surgiu na literatura, MRCMPSP (*Multi-Mode Resource-Constrained Multi-Project Scheduling Problem*), esse problema é foco principal do presente trabalho e foi proposto pela MISTA 2013.

4.2 Tipos de Recursos

Um projeto necessita de recursos para que as atividades possam ser executadas. Três categorias de recursos são definidas na literatura [Kolisch (1996)]:

- recursos renováveis (*renewable resources*): as quantidades disponíveis são renovadas de período em período (hora, dia, semana, mês), a disponibilidade por período é constante - exemplo: mão de obra, máquinas;
- recursos não-renováveis (*nonrenewable resources*): temos um consumo limitado dos recursos não-renováveis para todo o projeto - exemplo: dinheiro, energia e matéria prima;
- recursos duplamente limitados (*doubly constrained resources*): são recursos que são limitados no total do projeto e também por período - exemplo: dinheiro é um recurso limitado ao total do projeto, mas caso tenha um fluxo de caixa limitado por período ele será duplamente limitado.

Quando trata-se de escalonar múltiplos projetos, mais uma categoria de recurso é apresentada pela MISTA 2013:

- recursos globais (*global resources*): são recursos da categoria renováveis, porém as quantidades disponíveis são compartilhadas entre todos os projetos.

4.3 Função Objetivo

O objetivo proposto pela MISTA 2013, é encontrar uma solução factível, que minimize o TPD (*Total Project Delay*) e o TMS (*Total Makespan*).

- *total project delay*: é a soma das diferenças entre a *Critical Path Duration* (CPD) e o *makespan* atual de cada projeto;
- *total makespan*: é dado pelo maior tempo final menos o menor tempo de início entre os projetos.

4.4 Restrições

O PSP envolve a atribuição de atividades em determinados instantes de tempo em um horizonte, levando em consideração algumas restrições fortes, que por definição devem ser satisfeitas obrigatoriamente.

Neste trabalho consideram-se as seguintes restrições fortes:

- restrições de precedências: onde uma atividade j não pode iniciar até que todas as atividades antecessoras a ela tenham sido finalizadas;
- restrições de recursos: onde para executar uma atividade j utilizam-se recursos que possuem capacidades limitadas e as mesmas não podem ser extrapoladas.

5 Trabalhos Relacionados

Seis métodos como referência para o RCPSP *Resource-Constrained Project Scheduling Problem* são apresentados em [Kolisch (2005)]:

[Alcaraz (2004)] desenvolveu um algoritmo genético com base na representação de lista de atividades. É utilizado um gene para a indicação de qual SGS (*Schedule Generation Scheme*) será utilizado, o SGS define a maneira em que a programação das atividades será gerada, detalhes são explicados na seção 6.2.1. Um gene adicional indica qual dos métodos (*forward* ou *backward*) será utilizado para fazer o deslocamentos nos instantes de tempo das atividades contidas na lista. O operador de crossover para listas de atividades se estende de tal forma que a lista de atividades pode ser construída de forma bidirecional que programa as atividades combinando *forward* e *backward*, ou seja, programa as atividades tanto para frente como para trás. Nessa abordagem aplica-se o FBI (*Forward-Backward Improvement*), método que será explicado na seção 6.3.

[Debels (2004)], faz uso de abordagens baseadas em população, onde aplica pesquisa de dispersão baseado na população, uma variante dos algoritmos genéticos. Ele utiliza SGS em série e faz uso da representação topológica, que é a ordenação dos nós de forma linear, aonde cada nó vem antes dos nós ligados às suas arestas de saída. Foi definido um novo operador de recombinação que consiste na combinação linear das soluções. Essa abordagem faz o uso do método FBI.

[Hartmann (2002)], propõe um algoritmo genético de auto-adaptação. Esta abordagem aumenta a representação da lista de atividade pela adição de um gene que determina se o SGS em série ou paralelo será usado na lista de atividades. Como pré-requisito para o procedimento, nessa abordagem o SGS paralelo é responsável pela decodificação da lista de atividade. Os melhores esquemas gerados são herdados e continuam na população de sobreviventes do algoritmo genético.

[Kochetov (2003)], elaborou um algoritmo evolutivo que combina o algoritmo genetico, reconexão de caminhos, e busca tabu. As soluções são evoluídas escolhendo duas soluções e construindo um caminho através das conexões entre as selecionadas. A melhor solução do caminho é escolhida e melhorada através da busca tabu. A melhor solução a partir da busca tabu é adicionada à população e a pior solução é removida da população.

[Vallsa (2005)] introduz vários esquemas baseados em população não-padrão em um estudo que incide sobre o FBI. A população não-padrão é a diferença na forma como os pais são selecionados para a reprodução e na quantidade de descendentes produzidos

por cada par de pais. O operador de mudança substitui posições aleatórias do primeiro progenitor com as posições correspondentes ao segundo progenitor.

[Tormos (2001)], aplicou FBI para melhorar as programações construídas por metaheurísticas ou *X-pass*, que é uma heurística construtiva que combinam regras de prioridade e SGS para construir soluções[Kolisch (1998)]. Nesse procedimento as atividades são simplesmente deslocadas para a direita dentro da programação e, em seguida, para a esquerda, esse método produz excelentes resultados e pode ser combinado com quase qualquer outra abordagem.

Quatro das seis melhores abordagens citadas anteriormente fazem uso do *Forward-Backward Improvement*, o uso constante desse método e as melhorias nas soluções obtidas através dele o tornará um método importante e significativo nas heurísticas para o PSP.

Em relação às metaheurísticas os algoritmos genéticos e busca tabu foram as estratégias mais populares [Kolisch (2005)].

6 Heurísticas

Os procedimentos heurísticos em geral são divididos em duas categorias: heurísticas construtivas e heurísticas de busca local. Heurísticas construtivas começam a partir do zero e adicionam atividades, uma a uma, até que uma programação factível, detalhada na seção 6.1, seja obtida. As atividades são normalmente programadas usando regras de prioridade, detalhadas na seção 6.2.2. Heurísticas de busca local melhoram uma programação factível, obtida por alguma heurística construtiva.

Antes de apresentar as diferentes heurísticas construtivas e de busca local é necessario apresentar alguns detalhes sobre os tipos de *schedules*, em português será denominado programação.

Para ilustrar os procedimentos heurísticos, vamos considerar a rede apresentada na figura 7, sobre cada nó são indicadas as informações do tempo de execução e o consumo de recursos de cada atividade, neste exemplo é considerado apenas um modo de execução. Para conseguir representar todos os tipos de *schedules*, trocaremos a duração da atividade 3 mostrada na figura para 1 instante.

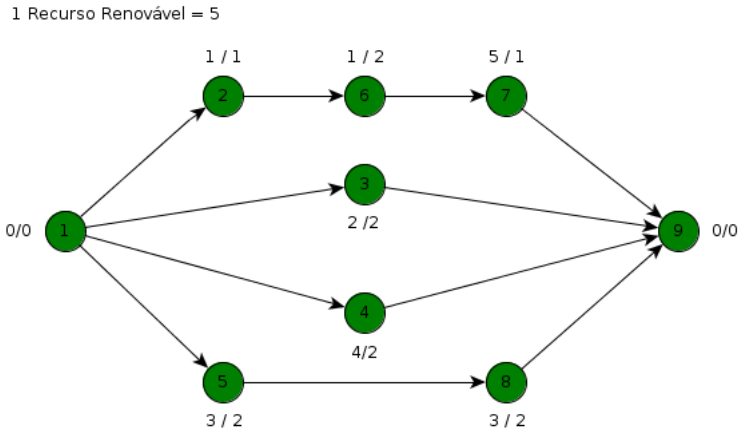

Figura 7. Exemplo de um problema para ilustrar os procedimentos heurísticos

6.1 Tipos de Programação

Os diferentes tipos de programação que são apresentados, facilitam a apresentação e entendimento dos procedimentos heurísticos nas próximas seções. São eles:

- programação factível (*feasible schedule*): é uma programação viável, que consiste em ser completa e nenhuma das restrições de precedência ou de recursos podem estar violadas — uma programação é completa quando todas as atividades j do projeto são alocadas em algum tempo de início — uma programação viável para o problema da figura 7 é apresentado na figura 8 obtida pela atribuição dos seguintes horários de início para cada atividade: $1 \rightarrow 0$, $2 \rightarrow 0$, $3 \rightarrow 7$, $4 \rightarrow 4$, $5 \rightarrow 0$, $6 \rightarrow 2$, $7 \rightarrow 3$, $8 \rightarrow 3$ e $9 \rightarrow 8$;

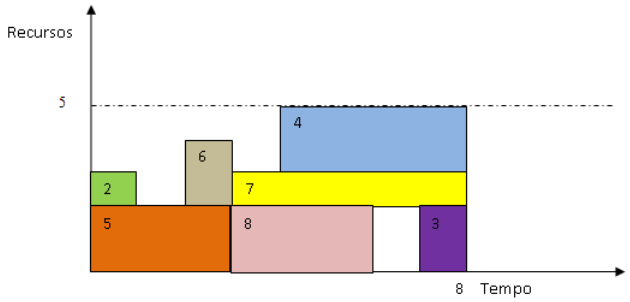

Figura 8. ***Feasible schedule***

- programação semi-ativa (*semi-active schedule*): uma programação semi-ativa é obtida através de deslocamentos locais à esquerda das atividades, até que nenhuma outra mudança local à esquerda possa ser realizada — o operador de deslocamento local corresponde ao decremento repetitivo unitário no tempo inicial de uma atividade — a figura 9 apresenta uma programação semi-ativa para o problema apresentado na figura 7 obtida pelos deslocamentos locais à esquerda das atividades 3, 4, 6 e 7;

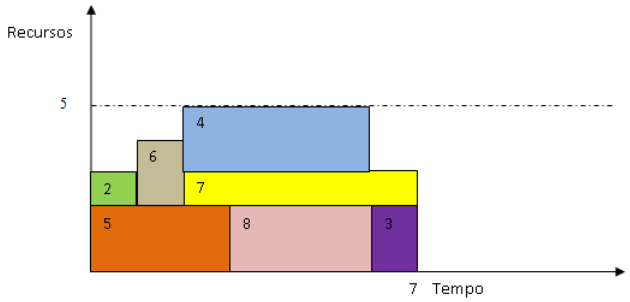

Figura 9. ***Semi-active schedule***

- programação ativa (*active schedule*): uma programação ativa é obtida quando não há mais deslocamentos globais ou locais para a esquerda a serem realizados — a redução do tempo de início de uma atividade em pelo menos um instante, de forma a não violar as restrições de recursos é chamada de deslocamento global à esquerda — a figura 10 apresenta uma programação ativa para o problema da figura 7 obtida pelo deslocamento global à esquerda da atividade 3;

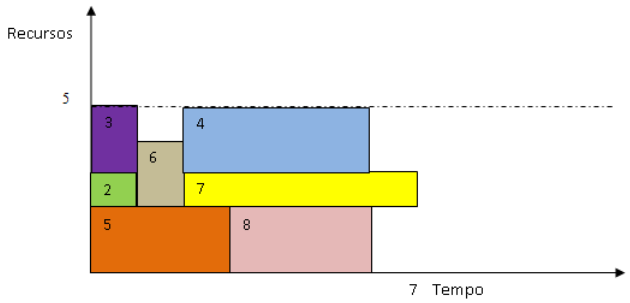

Figura 10. ***Active schedule***

- programação sem atraso (*non-delay schedule*): é uma programação factível onde não existe qualquer período de tempo tal que uma atividade elegível poderia ter sido alocada no início do período e não foi — uma atividade é elegível se não viola relações de precedências e nem restrições de recursos em um dado instante — a figura 11 representa uma programação sem atraso para o problema da figura 7.

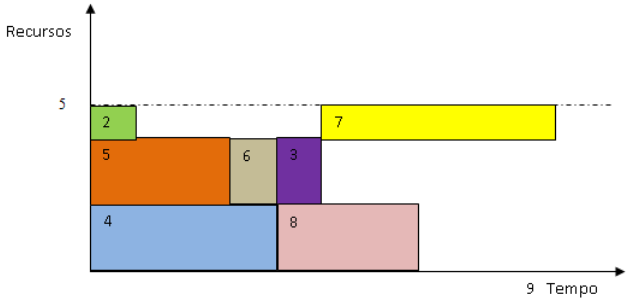

Figura 11. ***Non-delay schedule***

6.2 Heurísticas Construtivas

Nesta seção são apresentadas heurísticas que têm como objetivo construir uma solução factível para o PSP. Estas heurísticas utilizam esquemas de programação e regras de prioridade.

Os esquemas de programação determinam a maneira como a programação será construída através da atribuição de tempos de início para as atividades. Dois esquemas para geração de programas são apresentados, são eles, em série e paralelo.

As regras de prioridade determinam as atividades que serão selecionadas durante o processo de busca da heurística.

6.2.1 Esquemas de Programação

Para ilustrar os diferentes tipos de esquemas existentes, será levada em consideração a lista de prioridade <1,2,6,5,7,4,8,3,9> obtida pela ordem das atividades da figura 7.

Esquema de Geração em Série

O SGS (*Schedule Generation Scheme*) em série, acrescenta sequencialmente uma atividade à programação até que a mesma esteja completa e factível, conceitos introduzidos na seção 6.1. A cada iteração, a próxima atividade da lista de prioridades é escolhida e atribuída ao primeiro tempo de início possível, de tal forma que não viole nenhuma restrição de precedência ou recurso. Para gerar um esquema em série é preciso:

- escolher a atividade de acordo com a prioridade;
- alocar a atividade no menor instante de tempo viável;
- apenas uma atividade pode ser alocada a cada iteração;
- possuir dois conjuntos disjuntos, S_g de atividades já programadas e D_g de atividades elegíveis — uma atividade é elegível se ela não viola nenhuma restrição de precedência ou recursos.

Aplicando o esquema de geração em série na lista de prioridades apresentada no início da seção, é obtida uma programação factível apresentada na figura 12 com um *makespan* de 8 unidades.

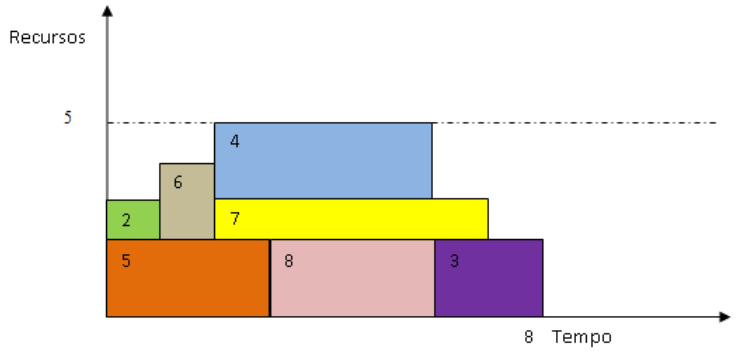

Figura 12. Programação Esquema de Geração em Série

Primeiramente são atribuídos às atividades 1, 2, 6, 5 e 7 tempos iniciais baseados nas relações de precedências. A próxima atividade da lista de prioridades é a 4, com base na relação de precedência, a atividade 4 poderia começar no instante 0, porém ocorrerá um conflito de recursos no período 2 com as atividades 5 e 6. Como resultado a atividade 4 poderá começar no instante 2. De modo semelhante, para as atividades 8 e 3 são atribuídos tempos de início 3 e 6 respectivamente, respeitando as restrições de precedência e de recursos.

Esquema de Geração Paralelo

O SGS (*Schedule Generation Scheme*) paralelo, ao contrário do esquema de geração em série, faz iterações sobre diferentes pontos de decisão nos quais as atividades poderão ou não ser adicionadas à programação.

Estes pontos de decisão correspondem aos tempos de conclusão das atividades já programadas e, portanto, os n pontos de decisão precisam ser considerados. Em cada ponto de decisão, as atividades não programadas, cujas atividades antecessoras tenham sido concluídas, são consideradas em ordem de prioridade na lista e alocadas analisando

as restrições de recursos naquele instante de tempo. Para gerar um esquema paralelo é preciso:

- a cada iteração, alocar várias atividades elegíveis com início menor ou igual a t_g sobre um ponto de decisão, baseado nas prioridades — caso não seja possível em relação aos recursos, será considerada nas próximas iterações;
- cada iteração g possui um t_g associado, que corresponde ao tempo mínimo das atividades já alocadas;
- o conjunto D_g corresponde às atividades elegíveis para começar em t_g.

Aplicando o esquema de geração paralelo na lista de prioridades apresentada no início da seção, é obtida uma programação factível apresentado na figura 13, com um *makespan* de 9 unidades.

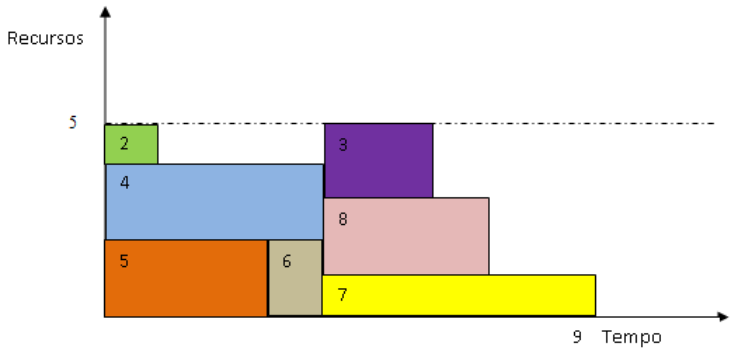

Figura 13. Programação Esquema de Geração em Paralelo

Após a atividade de início ser agendada no instante 0, quatro atividades 2, 3, 4 e 5 serão selecionadas com o ponto de decisão 0. A atividade 2 é a primeira da lista de prioridades e será programada no instante 0. A atividade 5 vem em seguida na lista e também não causará nenhum conflito de recursos no período 1, logo ela será iniciada no instante 0. A atividade 4 é a próxima na lista e como existem recursos suficientes, ela também será programada no instante 0. A atividade 3 é a última da lista e também não há mais recursos suficientes no período 1, por isso a atividade 3 permanece.

O próximo ponto de decisão é o instante 1, quando a atividade 2 é concluída. Neste instante as atividades 3 e 6 são as únicas não programadas e cujo antecessores já foram concluídos, a atividade 6 vem antes da 3 na lista de prioridades, como não há recursos suficientes para agendar nenhuma delas precisamos para passar para o próximo ponto de decisão, que é o instante 3, instante no qual a atividade 5 é concluída. As atividades programadas cujos antecessores foram completados são as atividades 6, 8 e 3 em ordem de prioridade. Somente a atividade 6 poderá começar no instante 3 em relação aos recursos.

O próximo ponto de decisão será o instante 4, onde as atividades 4 e 6 são concluídas, neste momento teremos as atividades 7, 8 e 3, em ordem de prioridade para serem agendadas no instante 4, visto que não há conflito de recursos. Depois da conclusão a atividade 9 poderá ser agendada no ponto de decisão 9.

6.2.2 Regras de Prioridades

As abordagens básicas para as heurísticas construtivas explicadas utilizam listas de prioridades.

Estas regras de prioridade podem ser classificadas em cinco categorias apresentadas em [Demeulemeester (2002)], com base no tipo de informação que é necessário para calcular a lista de prioridades. São elas:

- regras de prioridade baseadas em atividades: consideram as informações que estão relacionadas com as atividades e não com o projeto;
- regras de prioridade baseadas em rede: consideram apenas as informação que estão contidas na rede — nenhuma informação sobre recursos podem ser utilizadas para as regras de prioridade nesta categoria;
- regras de prioridade baseadas no caminho crítico: são baseadas nos cálculos do caminho crítico;
- regras de prioridade baseadas em recursos: consideram o uso de recursos das diferentes atividades;
- regras de prioridade baseadas em composição: podem ser obtidas através da soma ponderada dos valores de prioridade obtidos nas regras das três categorias anteriores.

A regra de prioridade mais utilizada na literatura para o PSP é a baseada no caminho crítico, por isso ela será detalhada nesta seção.

A terceira categoria, regra de prioridade baseada no caminho crítico, possui seis regras de prioridades estáticas, regras estáticas precisam ser calculadas no início do processo de programação, são elas EST (*Earliest Start Time*), EFT (*Earliest Finish Time*), LST (*Latest Start Time*), LFT (*Latest Finish Time*), MSLK (*Minimum Slack*) e RSM (*Resource Scheduling Method*) e três regras de prioridade dinâmicas, que dependem de uma programação parcial para poder atualizar os tempos das atividades, são elas ESTD (*Dynamic Earliest Start Time*), EFTD (*Dynamic Earliest Finish Time*) e MSLKD (*Dynamic Minimum Slack*).

Abaixo são apresentadas as listas de prioridade geradas para as regras mais utilizadas na literatura, em relação ao problema da figura 7.

- EST: $<$ 1,2,3,4,5,6,7,8,9 $>$
- EFT: $<$ 1,2,3,6,5,4,8,7,9 $>$
- LST: $<$ 1,2,5,6,7,4,8,3,9 $>$
- LFT: $<$ 1,2,6,5,3,4,7,8,9 $>$

6.3 Heurística de Busca Local

Um método que vem sendo muito utilizado atualmente para o PSP é o FBI *(Forward-Backward Improvement)*, esse método faz dupla justificação na programação das atividades [Vallsa (2005)] e produz um aperfeiçoamento de uma programação, o método é composto por dois passos.

Na primeira etapa *(forward)*, as atividades são alinhadas à direita no período de tempo, isto é, exceto para as atividades fictícias, todas as alocações das atividades são

deslocadas para a direita a partir dos predecessores imediatos da atividade final, até sucessores imediatos da atividade inicial. Esta etapa gera uma programação ativa para a direita (um programa em que nenhuma atividade pode ser concluída mais tarde, sem avançar qualquer outra atividade, ou violar as restrições, ou aumentar o *makespan*) [Vallsa (2005)]. A atividade final não é deslocada, assim, o *makespan* atual é mantido. Se durante a primeira etapa for gerada uma folga para a programação, o segundo passo tentará reduzir o *makespan* retirando esta folga, como descrito no parágrafo seguinte.

Na segunda etapa *(backward)*, as atividades são analogamente justificadas à esquerda da programação, isto é, com exceção da atividade fictícia inicial, todas as alocações das atividades são deslocadas para a esquerda, a partir dos sucessores imediatos da atividade inicial, até a atividade final. Esta etapa gera uma programação ativa (um programa em que nenhuma atividade pode ser iniciada mais cedo, sem atrasar alguma outra atividade, ou violar as restrições).

Em ambas as etapas a programação é sempre viável, uma vez que as relações de precedências e as restrições de recursos são satisfeitas. Cada etapa é realizada usando o esquema de geração em série explicado na seção 6.2.1.

7 Programação Inteira

Nesta seção é apresentada uma formulação de Programação Inteira (PI) para o PSP proposta em [Kolisch (1996)]. Esse modelo auxilia na compreensão do problema, visto que contempla todas as restrições impostas, no entanto, não é capaz de resolver instâncias complexas em um tempo viável.

Assim, foi produzido um algoritmo híbrido que contempla vários modelos baseados em PI para otimização de subproblemas. São eles: construtivo, modo mínimo (*ModeMin*) e busca local, esses modelos propostos em [Toffolo (2013)] correspondem à abordagem para a construção do resolvedor enviado à competição MISTA 2013, em andamento. São apresentados nas seções subsequentes.

A seguir são definidos os dados de entrada do modelo proposto em [Kolisch (1996)]:

J : conjunto de atividades;
M_j : modos em que a atividade j pode ser executada;
$R(N)$: tipos de recursos disponíveis, recursos renováveis e não renováveis;
$K_r^p \geq 0$: número de recursos renováveis por período;
$K_r^v \geq 0$: número de recursos não renováveis;
$k_{jmr}^p \geq 0$: quantidade de recursos do tipo renovável necessários para a atividade j ser executada no modo m;
$k_{jmr}^v \geq 0$: quantidade de recursos do tipo não renovável necessários para a atividade j ser executada no modo m;
P_j : atividades predecessoras da atividade j;
EST_j : tempo do início da janela da atividade j;
LFT_j : tempo do fim da janela da atividade j.

A variável de decisão é indexada por três índices binários que representam respectivamente o job, modo e tempo de término da alocação.

x_{jmt} : variável binária: 1 se a atividade j é alocada no modo m e termina no instante t, 0 caso contrário.

Minimize

$$\sum_{m=1}^{M_J} \sum_{t=EST_J}^{LFT_J} t.x_{Jmt} \tag{1}$$

Sujeito a

$$\sum_{m=1}^{M_j} \sum_{t=EST_j}^{LFT_j} x_{jmt} = 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{m=1}^{M_h} \sum_{t=EST_h}^{LFT_h} t.x_{hmt} \leq \sum_{m=1}^{M_j} \sum_{t=EST_j}^{LFT_j} (t - d_{jm}) x_{jmt} \qquad \forall j \in j = 2, ..., J, \forall h \in P_j \tag{3}$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} K_{jmr}^{p} \sum_{q=max(t,EST_j)}^{min(t+d_{jm}-1,LFT_j)} x_{jmq} \leq K_r^p \qquad \forall r \in R, \forall t \in T \tag{4}$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} K_{jmr}^{v} \sum_{t=EST_j}^{LFT_j} x_{jmt} \leq K_r^v \qquad \forall r \in N \tag{5}$$

$$\tag{6}$$

7.1 Modelo Modo Mínimo

O principal obstáculo para a produção de soluções viáveis para o MRCMPSP são os recursos não-renováveis. Embora o uso de recursos renováveis possa implicar no atraso ou aceleração de projetos, o uso excessivo de recursos não-renováveis na alocação de algumas atividades pode facilmente inviabilizar a alocação de outras. Uma opção é a definição dos modos das atividades com antecedência para que elas respeitem o uso de recursos não-renováveis. Denotamos o problema de selecionar estes modos iniciais como *ModeMin*.

Neste problema queremos definir para cada atividade um modo, tal que o limite total de consumo dos recursos não-renováveis seja sempre respeitado e as durações das atividades sejam minimizadas. O *ModeMin* é tão difícil quanto o problema da mochila m-dimensional, para isso utilizaremos PI (Programação Inteira) para definir estes modos.

A *ModeMin* dá uma estimativa válida do consumo de recursos não-renováveis para as atividades que não foram alocadas e orienta o nível de ganância das heurísticas de programação construtiva.

A seguir são definidos os dados de entrada:

J : conjunto de atividades;
P : conjunto de projetos;
C : conjunto de caminhos;

K : conjunto de recursos não renováveis;
j : atividade j;
t : tempo t;
m : modo m;
p : projeto p;
c : caminho c;
k : recurso não renovável k;
d_{jm} : duração da atividade j no modo m;
q_{kjm} : quantidade demandada do recurso não renovável k para a atividade j no modo m;
q_k^{disp} : quantidade disponível do recurso k;

A seguir são definidas as variáveis de decisão:

x_{jm} : variável binária: 1 se a atividade j foi executada no modo m, 0 caso contrário;
z_p : variável inteira: define o fim dos projetos;

Minimize

$$\begin{cases} F_1 = \sum_{j \in J} \sum_{m \in M} x_{jm} \\ F_2 = \sum_{p \in P} z_p \end{cases} \tag{7}$$

Sujeito a

$$\sum_{m \in M} x_{jm} = 1 \qquad \forall j \in J \tag{8}$$

$$\sum_{j \in J} \sum_{m \in M} (q_{kjm}\, x_{jm}) \leq q_k^{disp} \qquad \forall k \in K \tag{9}$$

$$z_p \geq \sum_{c \in C} \sum_{j \in J} \sum_{m \in M} d_{jm}\, x_{jm} \qquad \forall p \in P \tag{10}$$

7.2 Modelo Construtivo PI

Uma heurística PI é usado para construir uma solução factível inicial. Na construção a seleção dos modos é feita garantindo que a utilização de recursos não-renováveis para as atividades alocadas não implicará na falta desses recursos para alocação das atividades remanescentes em seus respectivos *ModeMins*.

A cada iteração uma janela de tempo é definida, iniciando-se no instante zero. Em cada janela o modelo PI busca alocar um número máximo de atividades considerando as restrições originais, como relação de precedências e consumo de recursos. Estas janelas de tempo são sequenciais e não-sobrepostas.

O algoritmo termina quando todos as atividades são alocados.

A seguir são definidos os dados de entrada:

J : conjunto de atividades;

P : conjunto de projetos;

K : conjunto de recursos não renováveis;

R : conjunto de recursos renováveis;

T : tamanho máximo da janela;

j : atividade j;

t : tempo t;

m : modo m;

p : projeto p;

k : recurso não renovável k;

r : recurso renovável r;

d_{jm} : duração da atividade j no modo m;

q_{rjm} : quantidade demandada do recurso r para a atividade j no modo m;

q_k^{disp} : quantidade disponível do recurso k;

q_r^{disp} : quantidade disponível do recurso r;

EST_{jm} : o menor início da atividade j no modo m;

LFT_{jm} : o maior fim da atividade j no modo m;

usK_k : quantidade do recurso não renovável k usados;

usR_{tr} : quantidade do recurso renovável r usados no instante t;

fuK_k : quantidade futura necessária para o recurso não renovável k;

cM : constante de peso;

cW : constante de peso;

A seguir são definidas as variáveis de decisão:

x_{jmt} : variável binária: 1 se a atividade j foi executada no modo m e iniciou em t, 0 caso contrário;

y_p : variável inteira: define o fim dos projetos;

$yMax$: variável inteira: define o maior fim entre os projetos;

Maximize

$$\begin{cases} F_1 = \sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} \\ F_2 = -\sum_{p\in P} y_p \\ F_3 = -yMax \end{cases} \tag{11}$$

Sujeito a

$$\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} \leq 1 \qquad \forall j \in J \tag{12}$$

$$\sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} (q_{kjm}\, x_{jmt}) + \sum_{j\in J}\left(1 - \sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt}\right) q_{kj\tilde{m}_j} \leq q_k^{disp} - usK_k - fuK_k \qquad \forall k \in K \tag{13}$$

$$\sum_{j\in J}\sum_{m\in M}\sum_{q=max(t-d_{jm}+1,EST_{jm})}^{min(t,LFT_{jm})} (q_{rjm}\, x_{jmq}) \leq q_r^{disp} - usR_{tr} \qquad \forall r \in R, \forall t \in T \tag{14}$$

$$\sum_{m\in M}\sum_{t=EST_{j_1m}}^{LFT_{j_1m}} (t + d_{j_1m})\, x_{j_1mt} - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} t\, x_{j_2mt} - \left(1 - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} x_{j_2mt}\right) cM \leq 0 \qquad \forall (j_1, j_2) \in Pred \tag{15}$$

$$\sum_{m\in M}\sum_{t=EST_{j_1m}}^{LFT_{j_1m}} x_{j_1mt} - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} x_{j_2mt} \geq 0 \qquad \forall (j_1, j_2) \in Pred \tag{16}$$

$$y_p \geq \left(1 - \sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt}\right) cW \qquad \forall p \in P, \forall j \in J \tag{17}$$

$$yMax - y_p \geq 0 \qquad \forall p \in P \tag{18}$$

7.3 Modelo Busca Local PI

A busca local é um modelo similar à construção da solução inicial, visto que também é necessário ter uma janela para criar os subproblemas a serem resolvidos a partir da solução inicial.

Uma janela de tempo é definida e apenas as atividades da solução corrente que estão neste intervalo serão elegíveis para alterar os modos e os tempos iniciais. Este procedimento de busca local tem o objetivo de alterar os modos e a organização temporal da solução, de modo que se obtenha uma solução onde as tarefas em questão são concluídas antes ou, ao menos, uma configuração diferente da atual é definida, o que pode permitir resultados melhores em otimizações subsequentes.

A busca local baseada em PI é combinado com o procedimento *Forward-Backward Improvement* (FBI) ou dupla justificação [Vallsa (2005)], descrito nas seção 5.3.

Após a aplicação do *Forward-Backward Improvement*, algumas atividades podem não ter sido deslocadas em quaisquer direções e podem representar um gargalo para o problema. Propomos um terceiro passo de melhoria que fixa estas atividades e muda aleatoriamente o modo de cada uma, tentando executar um deslocamento à esquerda usando o esquema de geração em série explicado na seção 6.2.1.

Estas três etapas são realizadas ao mesmo tempo para obter uma melhoria na qualidade da solução enquanto o limite de tempo não é alcançado.

A seguir são definidos os dados de entrada:

J : conjunto de atividades;
P : conjunto de projetos;
K : conjunto de recursos não renováveis;
R : conjunto de recursos renováveis;
T : tamanho máximo da janela;
j : atividade j;
t : tempo t;
m : modo m;
p : projeto p;
k : recurso não renovável k;
r : recurso renovável r;
d_{jm} : a duração da atividade j no modo m;
q_{rjm} : quantidade demandada do recurso r para a atividade j no modo m;
EST_{jm} : o menor início da atividade j no modo m;
LFT_{jm} : o maior fim da atividade j no modo m;
usK_k : quantidade do recurso não renovável k usados;
usR_{tr} : quantidade do recurso renovável r usados no instante t;

A seguir são definidas as variáveis de decisão:

x_{jmt} : variável binária: 1 se a atividade j foi executada no modo m e iniciou em t, 0 caso contrário;

Minimize

$$\sum_{j \in J} \sum_{m \in M} \sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} \tag{19}$$

Sujeito a

$$\sum_{m \in M} \sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} = 1 \qquad \forall j \in J \tag{20}$$

$$\sum_{j \in J} \sum_{m \in M} \sum_{t=EST_{jm}}^{LFT_{jm}} (q_{kjm}\, x_{jmt}) \leq q_k^{disp} - usK_k \qquad \forall k \in K \tag{21}$$

$$\sum_{j \in J} \sum_{m \in M} \sum_{q=max(t-d_{jm}+1, EST_{jm})}^{min(t, LFT_{jm})} (q_{rjm}\, x_{jmq}) \leq q_r^{disp} - usR_{tr} \qquad \forall r \in R, \forall t \in T \tag{22}$$

$$\sum_{m \in M} \sum_{t=EST_{j_1 m}}^{LFT_{j_1 m}} (t + d_{j_1 m})\, x_{j_1 mt} - \sum_{m \in M} \sum_{t=EST_{j_2 m}}^{LFT_{j_2 m}} t\, x_{j_2 mt} \leq 0 \qquad \forall (j_1, j_2) \in Pred \tag{23}$$

8 Engenharia de *Software* Baseada em Pesquisa

A aplicação de técnicas de otimização para problemas da engenharia de *software* tem ganhado crescente atenção da comunidade acadêmica, constituindo um novo campo denominado *Search-Based Software Engineering* - SBSE[3].

O gerenciamento de projetos é uma atividade fundamental aos projetos de *software*. Em [Pressman (2005)] é afirmado que projetos de *software* precisam ser gerenciados devido ao fato que desenvolver um *software* é uma atividade complexa, especialmente se o projeto envolve muitas pessoas trabalhando durante um tempo longo, ele afirma também que projetos de *software* precisam ser planejados e controlados pelo fato desta ser a única forma conhecida de gerir sua complexidade. Em [Kerzner (2009)] é afirmado que as responsabilidades mais importantes de um gerente de projeto é o planejamento, a integração e a execução de planos, muitas vezes priorizam o controle dos recursos e requerem um planejamento formal e detalhado.

Assim a alocação de recursos em projetos de *software*, é considerada uma atividade de suma importância, na qual pode ser tratada como o PSP.

[3] file:///tmp/repository.html

Atualmente alguns trabalho estão sendo propostos utilizando algoritmos genéticos (AGs) para resolver cenários de projetos de *software* diferentes. Algoritmo genético é uma metaheurística baseada nos mecanismos de seleção natural e da evolução genética, empregam busca aleatória em torno dos elementos com alta aptidão.

Em [Alba (2007)] os recursos considerados são pessoas com um conjunto de habilidades e um salário. Esses funcionários têm um grau máximo de dedicação ao projeto. Formalmente, cada funcionário é denotada com e_i, onde i varia de 1 a E (número de funcionários). É levado em consideração que SK é o conjunto de habilidades, e s_i a habilidade do funcionário i variando de 1 a $S = SK$. A habilidade do funcionário e_i é denotada como $e_i^{skill} \in SK$, o salário mensal como e_i^{salary} e a dedicação máxima ao projeto como e_i^{maxded}.

Tanto o salário quanto a dedicação máxima são números reais. O primeiro é expresso em unidades monetárias fictícias, enquanto o último é a razão entre a quantidade de horas dedicadas ao projeto e toda a extensão da jornada de trabalho do empregado. Um cenário é apresentado na figura 14.

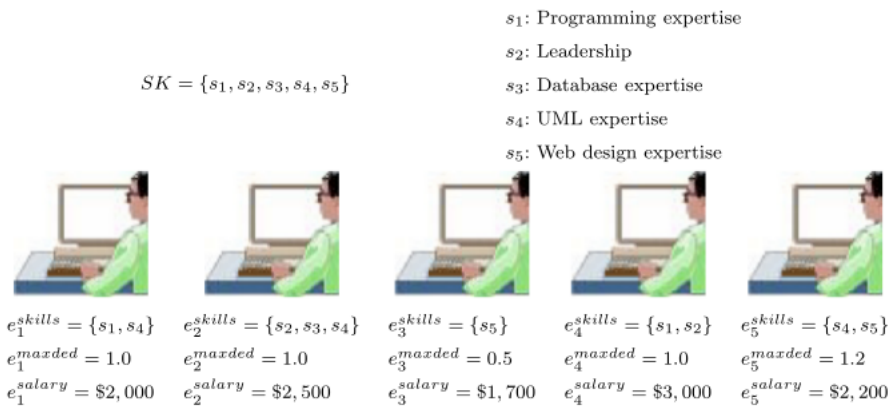

Figura 14. Cenário de uma empresa de *software*. [Alba (2007)]

Em [Alba (2007)] utilizou-se um AG para resolver o PSP, o primeiro passo foi criar um conjunto de soluções iniciais aleatoriamente e aplicou-se uma operação de cruzamento para combinar os conteúdos dos dois progenitores formando uma nova solução. Essa nova solução será modificada posteriormente pela operação de mutação, responsável por alterar os indivíduos. Nem todos os indivíduos participam da reprodução, pois foi utilizado seleção elitista, torneio binário, onde cada um dos pais é selecionado como o melhor dos dois indivíduos ao acaso. Os operadores foram aplicados de forma estocástica, sendo que cada um teve uma probabilidade associada de aplicação no ciclo iterativo. Os melhores indivíduos e a geração recém criada são combinados a fim de que as melhores soluções sejam retidas para a utilização no próximo passo do algoritmo.

A solução para esse problema em [Alba (2007)] é dada por uma matriz X com elementos $x_{ij} \in [0, 1]$, foi levado em consideração que nenhum funcionário faz horas extras, de modo que o máximo de dedicação de todos os funcionários é 1. Por esse motivo, o valor máximo para x_{ij} é 1. Foi utilizado cromossomos com sequência binária para representar as soluções dos problemas, logo foi preciso discretizar o intervalo $[0, 1]$ de modo a codificar o grau x_{ij} de dedicação. Três bits foi necessário para representá-los, na matriz X é armazenado um cromossomo x na linha principal, o comprimento do cromossomo é 3. A

figura 15 mostra a representação da solução.

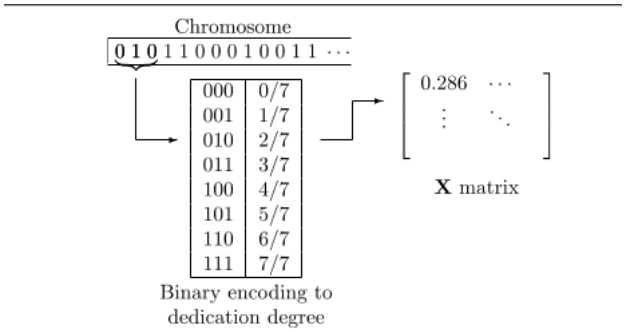

Figura 15. Representação da Solução do AG [Alba (2007)]

A função de avaliação para o cromossomo x apresentada em [Alba (2007)] é dada por:

$$f(x) \begin{cases} 1 \; se \; a \; solucao \; e \; factivel. \\ 0 \; caso \; contrario. \end{cases}$$

$$q = W_{custo} * P_{custo} + W_{dur} * P_{dur}$$

$$p = W_{penal} + W_{undt} * undt + W_{reqsk} * reqsk + W_{over} * P_{over}$$

Resumindo, a função de avaliação possui dois pesos, o custo da solução (q) e a penalidade de soluções inviáveis (p).

O primeiro termo é a soma ponderada do custo do projeto e duração. O W_{custo} e W_{dur} são valores de ponderação da importância relativa dos dois objetivos. Estes pesos permitem a aptidão para ser adaptado de acordo com as necessidades dos gerentes de projeto. Os gerente de projeto podem experimentar diferentes valores de pesos a fim de adaptar as soluções às suas necessidades.

A penalidade p é a soma ponderada dos parâmetros da solução que o tornam inviável, ou seja, o excesso de trabalho do projeto P_{over}, o número de atividades com nenhum funcionário associado $undt$, e o número de habilidades ainda necessários a fim de executar todas as atividades do projeto $reqsk$. Cada um destes parâmetros é pesado e adicionado à constante w_{penal}, responsável por separar a faixa aptidão das soluções viáveis partir das inviáveis. Cada um destes parâmetros é pesado e adicionado à constante w_{penal}, responsável por separar a faixa aptidão das soluções viáveis partir das inviáveis.

O trabalho [Alba (2007)] abordou *Projeto Scheduling Problem* com algoritmos genéticos a fim de encontrar boas soluções minimizando os custos e tempo dos projetos de *software*. Tem também como objetivo permitir ao gerente de *software* estudar diferentes cenários com uma ferramenta automática, a fim de tomar as decisões adequadas sobre o melhor projeto para sua empresa.

Referências

[Alba (2007)] Alba, E. and Chicano, J. F. (2007). Software project management with gas. *Information Science a International Journal*, 177:2380–2401.

[Alcaraz (2004)] Alcaraz, J., Maroto, C., and Ruiz, R. (2004). Improving the performance of genetic algorithms for the rcps problem. *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pages 40–43.

[Debels (2004)] Debels, D., De Reyck, B., Leus, R., and Vanhoucke, M. (2004). A hybrid scatter search: Electromagnetism metaheuristic for project scheduling. *European Journal of Operational Research*.

[Demeulemeester (2002)] Demeulemeester, E. L. and Herroelen, W. S. (2002). *PROJECT SCHEDULING: A Research Handbook*. Kluwer Academic Publishers, Leuven Belgium.

[Hartmann (2002)] Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, pages 433–448.

[Kerzner (2009)] Kerzner, H. (2009). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. 10a edicao edition.

[Kochetov (2003)] Kochetov, Y. and Stolyar, A. (2003). Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. *In Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*.

[Kolisch (1998)] Kolisch, R. and Hartmann, S. (1998). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. *Handbook on Recent Advances in Project Scheduling*.

[Kolisch (2005)] Kolisch, R. and Hartmann, S. (2005). Experimental investigation of heuristic for resource-constrained project scheduling problem: An update. *European Journal of Operational Research*.

[Kolisch (1996)] Kolisch, R. and Sprecher, A. (1996). Psplib - a project scheduling problem library. *European Journal of Operational Research*, 96:205–216.

[Pressman (2005)] Pressman, R. (2005). *Software Engineering: A Practitioners Approach. McGraw-Hill,*. 6a edicao edition.

[Slack (2013)] Slack, N., Chambers, S., Johnston, R., and Betts, A. (2013). *Gerenciamento de Operacoes e de Processos: Principios e pratica de impacto estrategico*. Bookman.

[Toffolo (2013)] Toffolo, T. A. M., Santos, H. G., Soares, J. A., and Carvalho, M. A. M. (2013). An integer programming approach for the multi-mode resource-constrained multi-project scheduling problem.

[Tormos (2001)] Tormos, P. and Lova, A. (2001). A competitive heuristic solution technique for resource constrained project scheduling. *Annals of Operations Research*, pages 65–81.

[Vallsa (2005)] Vallsa, V., Ballestina, F., and Quintanillab, S. (2005). Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165:375–386.