Universidade Federal do Rio de Janeiro

ENIO PIRES DE ABREU

UMA ABORDAGEM BRANCH & BOUND À MAXIMIZAÇÃO DO VALOR DE PROJETOS DE SOFTWARE EM AMBIENTES DE RECURSOS ESCASSOS

DISSERTAÇÃO DE MESTRADO

Instituto de Matemática

Núcleo de Computação Eletrônica

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ENIO PIRES DE ABREU

UMA ABORDAGEM BRANCH & BOUND À MAXIMIZAÇÃO DO VALOR DE PROJETOS DE SOFTWARE EM AMBIENTES DE RECURSOS ESCASSOS

RIO DE JANEIRO
2009

Enio Pires de Abreu

UMA ABORDAGEM BRANCH & BOUND À MAXIMIZAÇÃO DO VALOR DE PROJETOS DE SOFTWARE EM AMBIENTES DE RECURSOS ESCASSOS

Dissertação submetida ao corpo docente do Instituto de Matemática (IM) / Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Orientador: Prof. Eber Assis Schmitz, Ph.D.

Co-orientador: Prof. Antonio Juarez Sylvio Menezes de Alencar, D.Phil.

RIO DE JANEIRO
2009

A162 Abreu, Enio Pires de A..
Uma Abordagem branch & bound à maximização do valor de projetos de software em ambientes de recursos escassos / Enio Pires de Abreu. -- 2009.
69 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica, Rio de Janeiro, 2009.

Orientador: Eber Assis Schmitz
Co-orientador: Antonio Juarez Sylvio Menezes de Alencar

1. Planejamento de Projetos - Teses. 2. Branch And Bound - Teses. 3. Árvore de Busca - Teses. I. Eber Assis Schmitz (Orient.). II. Antonio Juarez Sylvio Menezes de Alencar (Co-orient.). III. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. IV. Título

CDD

Enio Pires de Abreu

UMA ABORDAGEM BRANCH & BOUND
À MAXIMIZAÇÃO DO VALOR DE
PROJETOS DE SOFTWARE EM
AMBIENTES DE RECURSOS ESCASSOS

Dissertação submetida ao corpo docente do Instituto de Matemática (IM) / Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Rio de Janeiro, 17 de Dezembro de 2009.

Prof. Eber Assis Schmitz, Ph.D.
IM/NCE - UFRJ
Orientador

Prof. Antonio Juarez Sylvio Menezes de Alencar, D.Phil.
IM - UFRJ
Co-orientador

Prof. Alexandre Correa, D.Sc.
UNIRIO

Prof. Gabriel Pereira da Silva, D.Sc.
IM - UFRJ

Prof. Márcio de Oliveira Barros, D.Sc.
UNIRIO

Dedicatória

À minha amada esposa Vanessa,
com respeito e admiração.

Agradecimentos

Agradeço a Deus e a todos aqueles que me ajudaram a realizar esta grande conquista.

Em especial ...

À minha amada esposa, Vanessa Silveira de Abreu, pelo seu carinho, apoio e incentivo que tornaram as dificuldades muito menores.

Aos meus orientadores, Antônio Juarez Alencar e Eber Assis Schmitz, pela orientação ativa e enriquecedora que tem se tornado cada vez mais rara nos cursos de mestrado.

Aos professores Gabriel Pereira da Silva e Adriano Joaquim de Oliveira Cruz, que me apoiaram em diversos momentos e contribuíram para a minha formação científica e pessoal.

Aos meus amigos Felipe Dias, Breno Peixoto e Rafael Alcemar, que contribuíram de diversas formas para a realização deste projeto.

À minha família, que sempre esteve ao meu lado, me ajudando a superar todas as dificuldades.

”Se existe uma forma melhor de fazer, descubra-a!”

Tomas Edison

Resumo

ABREU, Enio Pires de A.. **Uma Abordagem branch & bound à maximização do valor de projetos de software em ambientes de recursos escassos**. Rio de Janeiro, 2009. Dissertação (Mestrado em Informática) - Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2009.

A limitação de recursos, entre eles o humano e o financeiro, é um aspecto da realidade que influencia diretamente a forma como os projetos de software são desenvolvidos e, conseqüentemente, o retorno que este tipo de investimento pode fornecer. A partir do particionamento e da valoração do projeto em módulos mínimos de funcionalidades, e de uma descrição dos recursos disponíveis e dos recursos necessários para o seu desenvolvimento, é possível encontrar um plano para a execução desses módulos que maximize o valor presente líquido do projeto como um todo sem extrapolar a disponibilidade de recursos. Questões como o desenvolvimento em paralelo, a possibilidade de adiamento, recursos renováveis e não renováveis, limitações de tempo, e a possibilidade de utilizar os ganhos obtidos com os módulos prontos para ajudar no restante do projeto também podem ser consideradas para aprimorar ainda mais esse plano. Esta dissertação apresenta um método que determina o melhor plano de desenvolvimento para um projeto sob tais condições, maximizando seu valor para o negócio.

Palavras-chave: Planejamento de Projetos, Branch & Bound, Maximização do VPL Financeiro, Gerência de Projetos de Software.

Abstract

ABREU, Enio Pires de A.. **Uma Abordagem branch & bound à maximização do valor de projetos de software em ambientes de recursos escassos**. Rio de Janeiro, 2009. Dissertação (Mestrado em Informática) - Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2009.

Resource constraints, like human and financial, represent a reality aspect that directly affects the way software projects are developed and, consequently, the revenue that this kind of investment may return. After partitioning and valuating the project by means of self contained parts called Minimum Marketable Features, describing the available resources and evaluating the amount of resources that are required by each part, it is possible to find a development plan that maximizes the project overall Net Present Value without exceeding the resource availability. Considerations on parallel development, the delaying possibility, renewable and non renewable resources, time constraints, and the possibility of using the revenue from the ready parts of the project to fund the missing ones can also be used in order to improve the development plan returned by this method. This dissertation presents a method that builds the best development plan for a project under these conditions, maximizing its value to business.

Keywords: Project Planning, Branch & Bound, NPV maximization, Software Project Management.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AE	*Architectural Element*
FC	Fluxo de Caixa
FCD	Fluxo de Caixa Descontado
IFM	*Incremental Funding Method*
LI	Limite Inferior
LS	Limite Superior
MMF	*Minimum Marketable Feature*
TI	Tecnologia da Informação
VPL	Valor Presente Líquido
VPLC	Valor Presente Líquido de uma Cadeia

Sumário

1 Introdução

1.1 Contextualização do Problema

Nos mercados altamente competitivos e globalizados que marcam este início de século, é improvável que projetos de software cujos riscos não sejam conhecidos e aceitáveis para o negócio venham a ser sequer considerados para desenvolvimento (MCMANUS, 2003). Nestes mercados, os investidores clamam, cada vez mais e mais alto, por um rápido retorno de seus investimentos, períodos mais curtos para o desenvolvimento e comercialização de produtos, e uma arquitetura organizacional ágil e flexível (LAM, 2004; HELO; HILMOLA; MAUNUKSELA, 2004; WHITTLE; MYRICK, 2005). Todas essas necessidades requerem o uso de novas abordagens nos projetos de desenvolvimento de *software* e ferramentas capazes de reduzir custos, agilizar processos e melhorar a performance de produtos e serviços (BECK et al., 2001; JORGENSON; HO; STIROH, 2003; HIGHSMITH, 2002).

Segundo o relatório da Associação Brasileira das Empresas de Software, no ano de 2008 o mercado de software brasileiro movimentou mais de 5 bilhões de dólares (ABES, 2009). Paralelamente a todo esse investimento, temos que muitos projetos são cancelados ou encerrados prematuramente, mostrando que investir em software pode ser um negócio caro e muito arriscado (HUI; LIU, 2004). Uma das formas mais eficiente de reduzir o custo e o risco de um projeto de software é evitar construir tudo de uma única vez (modelo em cascata) (LARMAN, 2004; HIBBS; JEWETT; SULLIVAN, 2009). Além disso, para obter o comprometimento de todos e aumentar as chances de sucesso, o plano de um projeto deve considerar seu impacto financeiro para o negócio.

O desenvolvimento de software não é um custo que precisa ser reduzido com a prática do *outsourcing*, sendo, na verdade, uma atividade de criação de valor (TAYNTOR, 2007; AMIT; ZOTT, 2001); no entanto, engenheiros de sistemas geralmente não estão envolvidos ou não se interessam pelos objetivos empresariais de criação de valor. Isso porque o ensino e a prática do desenvolvimento de software se distanciaram em linguagem e valores daqueles que definem os requisitos desse software (LAPLANTE, 2007).

Muito é dito sobre como o software deve ser desenvolvido, mas pouco é dito sobre como o investimento deve ser realizado (LAPLANTE, 2007). Poucas empresas estão dispostas a esperar mais de um ano para ver o retorno de seus investimentos (LARMAN, 2004; DATTA, 2007).

Para lidar adequadamente com esta situação, tanto acadêmicos quanto desenvolvedores de *software* têm enfatizado a necessidade de métodos, conceitos e ferramentas que favoreçam a entrega rápida de funcionalidades que tenham valor para os clientes (ABACUS; BARKER; FREEDMAN, 2005; NORD; TOMAYKO, 2006). Neste contexto, uma abordagem de enfoque financeiro para a priorização de requisitos denominada *Incremental Funding Method* (Método de Construção Incremental), ou IFM, surgiu como uma forma de aumentar o valor potencial dos projetos de *software* (DENNE; CLELAND-HUANG, 2004a, 2004b).

Realizar o desenvolvimento de forma iterativa e no contexto da geração de valor, removendo ou mitigando o risco financeiro, é o objetivo do IFM. Para isso, as funcionalidades do *software* são agrupadas em unidades auto-contidas que criam valor para os negócios. O valor total trazido para uma organização por um *software* constituído de várias unidades interdependentes é fortemente influenciado pela ordem de implementação destas unidades, dado que cada uma possui seu próprio fluxo de caixa e restrições de precedência. Por este motivo, o método inclui um conjunto de estratégias de complexidade polinomial que auxiliam na elaboração de um plano de desenvolvimento que aumente o valor dos projetos, reduzindo os investimentos iniciais ou melhorando outras métricas de projeto tais como: o tempo necessário para se atingir o ponto de equilíbrio e o tempo de retorno do investimento (DENNE; CLELAND-HUANG, 2004b, 2005).

No entanto, o IFM é um método aproximativo e, portanto, nem sempre as estratégias identificadas pelo método levam ao melhor plano possível para o desenvolvimento das unidades de *software* que, no caso geral, só pode ser descoberto em tempo exponencial. Além disso, o IFM requer que cada unidade de *software* dependa de no máximo uma outra, para que o plano seja encontrado em tempo polinomial.

1.2 Definição do Problema

Particionar o projeto em módulos mínimos de funcionalidades nos permite avaliar o valor do projeto como um todo a partir da avaliação individual desses módulos. Contudo, as limitações de tempo, capital, pessoas e outros recursos são fatores que influenciam diretamente a estratégia de desenvolvimento e o retorno que os projetos de software podem oferecer.

Sendo assim, se para cada módulo tivermos uma descrição dos recursos necessários ao seu desenvolvimento, podemos encontrar um plano de desenvolvimento para o projeto que permita obter o maior Valor Presente Líquido possível diante das limitações dos recursos disponíveis.

O problema abordado nesta dissertação consiste em determinar o melhor plano de desenvolvimento para um projeto de software a partir do seu particionamento e valoração em termos de módulos mínimos de funcionalidades. Questões como o desenvolvimento em paralelo, a possibilidade de adiamento, recursos renováveis e não renováveis, limitações de tempo e a possibilidade de utilizar os ganhos obtidos com os módulos prontos para ajudar no restante do projeto também são consideradas na determinação desse plano. A descrição formal e a classificação do problema são apresentadas na Seção 2.2.

1.3 Contribuições

As principais contribuições deste trabalho são:

Frutos da pesquisa:

- Resultados parciais dessa pesquisa foram publicados em dois artigos, o primeiro no IV Simpósio Brasileiro de Sistemas de Informação (ALENCAR et al., 2008b) e o segundo no *Tenth International Conference on Enterprise Information Systems* (ALENCAR et al., 2008a).
- A programação do método foi realizada na linguagem Java, para a realização de testes e execução dos exemplos, facilitando a futura construção de uma ferramenta.

Benefícios da utilização do método:

- Para a área de TI é mostrar a atividade de desenvolvimento de software como uma oportunidade de geração de valor para o negócio.
- Para a área de gerência de projetos é mostrar como proceder para se obter o valor máximo que o projeto pode oferecer.
- Para o projeto é minimizar o impacto das restrições no resultado esperado, orientando a aplicação dos recursos disponíveis de forma a gerar o melhor resultado possível.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- Capítulo 2: Apresenta a revisão bibliográfica dos conceitos de gerência de investimentos em projetos de software utilizados na abordagem da proposta apresentada nesta dissertação e o método de Denne (DENNE; CLELAND-HUANG, 2004b) para determinar o plano de desenvolvimento, além de classificar o problema e o método proposto;
- Capítulo 3: Apresenta formalmente o método para determinar o melhor plano de desenvolvimento para a geração de valor;
- Capítulo 4: Apresenta um estudo de caso com a aplicação da proposta sugerida e discute os resultados encontrados;
- Capítulo 5: Discute e responde algumas perguntas importantes sobre o método proposto e seus benefícios, oferece perspectivas de trabalhos futuros e apresenta as considerações finais.

2 *Fundamentos Conceituais*

A seguir, é apresentada uma breve descrição do IFM. A descrição é baseada no livro (DENNE; CLELAND-HUANG, 2004b) e artigos de Denne e Cleland-Huang (DENNE; CLELAND-HUANG, 2004a, 2005). Logo depois, apresentamos a classificação do problema abordado e do método proposto, permitindo compará-lo com outros métodos similares.

2.1 O Método IFM

O IFM particiona os projetos de *software* em unidades auto-contidas denominadas *Minimum Marketable Features* (Pacotes Mínimos Comercializáveis), ou MMFs. Segundo (STEINDL, 2005; DENNE; CLELAND-HUANG, 2004b), esses pacotes agrupam funcionalidades que podem ser entregues rapidamente e que criam valor para os clientes (isto é, os clientes estão dispostos a pagar pelos serviços prestados por esses módulos) em uma ou mais das seguintes áreas:

- *Diferenciação competitiva* – A unidade de *software* habilita a criação de produtos ou serviços que são valorizados pelos clientes e que são diferentes de tudo que é oferecido no mercado;
- *Geração de lucro* – Embora a unidade de *software* não forneça nenhuma inovação que tenha valor para os clientes, ela aumenta o lucro propiciando o oferecimento de produtos de qualidade similar aos do mercado por preços melhores;
- *Redução de custos* – A unidade de *software* permite que a empresa economize dinheiro reduzindo os custos de execução de um ou mais processos;
- *Projeção da marca* – A construção da unidade de *software* faz com que a empresa projete uma imagem de ser tecnologicamente avançada; e
- *Aumento da fidelidade dos clientes* – A unidade de *software* faz com que os clientes comprem mais, com maior freqüência ou ambos.

Apesar de todo MMF ser auto-contido, são comuns os casos em que um MMF só pode ser desenvolvido depois que outras partes do projeto tenham sido concluídas. Estas outras partes podem ser outros

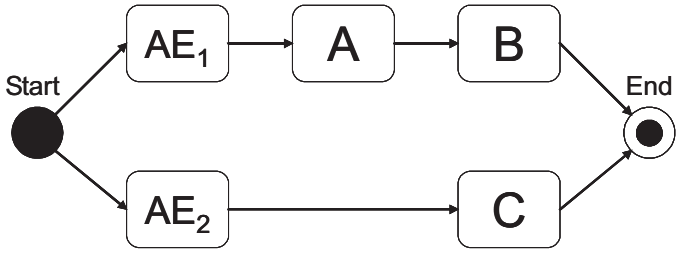

Figura 2.1: O grafo de precedência de um projeto.

MMFs ou a infra-estrutura arquitetural, isto é, o conjunto dos ítens básicos do projeto que não oferecem nenhum valor direto aos clientes, mas são requeridos pelos MMFs. Por exemplo, o software básico necessário para execução do projeto.

A própria arquitetura pode ser dividida em unidades auto-contidas passíveis de serem entregues separadamente. Estes elementos, chamados *Architectural Elements* (Elementos Arquiteturais) ou AEs, permitem que a arquitetura seja entregue conforme a necessidade, reduzindo o investimento inicial necessário para desenvolver o projeto. Leia (RASHID; MOREIRA; ARAúJO, 2003) para orientações sobre como obter os módulos e os elementos arquiteturais do software a partir dos requisitos.

2.1.1 Fluxo de caixa

Depois que os MMFs e os AEs são identificados, desenvolvedores e especialistas do negócio trabalham em conjunto para analisar cada MMF e AE, estimando os custos e os ganhos esperados para cada unidade no decorrer de uma janela de oportunidade. Veja uma discussão sobre como obter estas estimativas em projetos reais em (HUBBARD, 2007). Estes custos e ganhos constituem um fluxo de caixa que pode ser utilizado para estimar o valor total do *software*. Por exemplo, a Tabela 2.1 mostra o fluxo de caixa esperado para o conjunto de unidades do projeto apresentado na Figura 2.1.

Unidade de Projeto	**Período**							
	1	2	3	4	5	6	7	8
AE_1	-50	0	0	0	0	0	0	0
AE_2	-40	0	0	0	0	0	0	0
A	-20	35	35	35	35	35	35	35
B	-50	50	50	50	50	50	50	50
C	-40	30	30	30	30	30	30	30

Tabela 2.1: O fluxo de caixa de um conjunto de *MMFs* e *AEs* com valores em milhares de reais.

Na Tabela 2.1, os períodos são intervalos de tempo de mesma duração nos quais as despesas e receitas do projeto são apuradas. O tempo decorrido entre o primeiro e o último período é a janela de oportunidade do projeto, isto é, o período que se tem disponível para produzir um resultado. Além disso,

cada linha ao lado de uma unidade de projeto (MMF ou AE) é um fluxo de caixa, isto é, uma seqüência de despesas e receitas expressa em milhares de reais.

Podemos então observar que o MMF *A* requer um investimento inicial de 20 mil reais e, se pudesse ser desenvolvido no período 1, forneceria um ganho de 35 mil durante 7 períodos consecutivos. Mas isto não é possível, pois o desenvolvimento de *A* depende do desenvolvimento do elemento arquitetural AE_1. Conseqüentemente, *A* só pode ser desenvolvido a partir do período 2. Neste caso, o investimento inicial necessário seria o mesmo, mas o ganho de 35 mil só ocorreria durante 6 períodos. Portanto, em geral, quanto mais tarde o MMF for desenvolvido, menor será o ganho obtido.

Observe que todos os MMFs geram ganhos após um investimento inicial, enquanto que os AEs só apresentam custo no seu fluxo de caixa. Obviamente, o valor que um MMF traz para o projeto é fortemente afetado pelo comportamento do seu fluxo de caixa no decorrer do tempo.

Em termos formais, uma janela de oportunidade *P* é um conjunto $\{p_1, p_2, ..., p_k\}$ de períodos de mesma duração. Na Tabela 2.1, $P = \{1, 2, ..., 8\}$. O fluxo de caixa de uma unidade de projeto *v* é dado por $fc(v)$ e o elemento do fluxo de *v* no período $p \in P$ é dado por $fc(v, p)$. Na Tabela 2, $fc(A)$ no período 1, ou $fc(A, 1)$, é -20, e $fc(A)$ no período 2, ou $fc(A, 2)$, é 35.

2.1.2 Fluxo de caixa descontado

Para que se possa comparar os valores de diferentes MMFs no decorrer do tempo é preciso encontrar seus fluxos de caixa descontado (FCDs) já que não é apropriado executar operações matemáticas em valores monetários em diferentes instantes de tempo sem considerar uma taxa de juros (FABOZZI; DAVIS; CHOUDHRY, 2006). A soma de um FCD é o seu valor presente líquido, ou VPL. Em termos formais, o VPL de uma unidade de *software v*, cujo desenvolvimento inicia no período $t \in P$, considerando uma taxa de juros contante, é dado por

$$vpl(v,t) = \sum_{j=t}^{n} \frac{fc(v, j-t+1)}{(1+r)^j},$$

Onde *r* é a taxa de desconto e *n* é o último período da janela de oportunidade *P*. Por exemplo, se o desenvolvimento do MMF *A* começar no período 1, a uma taxa de desconto *r* de 2% por período, então seu VPL será

$$\begin{aligned} vpl(A,1) &= \frac{-20}{(1+\frac{2}{100})^1} + \frac{25}{(1+\frac{2}{100})^2} + \ldots + \frac{25}{(1+\frac{2}{100})^8} \\ &= \$202 \text{ mil} \end{aligned}$$

Mas se o desenvolvimento de *A* começar no período 2, então seu VPL será diferente, ou seja

$$\begin{aligned} vpl(A,2) &= \frac{-20}{(1+\frac{2}{100})^2} + \frac{35}{(1+\frac{2}{100})^3} + \ldots + \frac{35}{(1+\frac{2}{100})^8} \\ &= \$169 \text{ mil} \end{aligned}$$

A Tabela 2.2 mostra o VPL de cada MMF da Figura 2.1, considerando o início do desenvolvimento em diferentes pontos da janela de oportunidade *P*, a uma taxa de desconto de 2% por período, ou 2% pp. Note que a tabela só considera os cinco primeiros períodos da janela de oportunidade, isso porque é nestes períodos que as unidades de *software* serão construídas.

	Unidade de Projeto				
Períodos	**AE$_1$**	**AE$_2$**	**A**	**B**	**C**
1	-49	-39	202	268	151
2	-48	-38	169	221	123
3	-47	-38	137	175	96
4	-46	-37	105	130	69
5	-45	-36	73	85	42

Tabela 2.2: VPL do desenvolvimento das unidades do projeto considerando inicializações em diferentes períodos.

Repare que o valor do fluxo de caixa com desconto da unidade AE_1 no período 1 é -$49 mil e não -$50 mil. Isto se deve ao fato de que, embora os serviços comecem a ser realizados no início de um período, eles só são pagos ao final do período. Neste espaço de tempo, de acordo com a taxa de desconto, o dinheiro passa a valer um pouco menos (FABOZZI; DAVIS; CHOUDHRY, 2006).

2.1.3 A relação de precedência

As relações de dependência entre MMFs e AEs podem ser representadas por um grafo direcionado acíclico que restringe o desenvolvimento do projeto. Podemos observar um desses grafos na Figura 2.1.

No grafo, os MMFs são representados pelas letras *A*, *B* e *C*; enquanto os AEs são representados pelos elementos enumerados AE_1 e AE_2. Uma seta de *A* para *B*, isto é, $A \rightarrow B$, indica que o desenvolvimento da unidade A deve ser concluído antes que o de *B* comece. Os nós *Start* e *End* servem apenas para marcar o começo e o término do projeto, respectivamente, e por isso, possuem duração e fluxo iguais a zero.

Em termos formais, um grafo de precedência é uma estrutura matemática $G(V_G, E_G)$, na qual:

- $V_G = \{v_1, v_2, \cdots, v_n\}$ é um conjunto de MMFs e AEs, onde cada uma das unidades do *software* possui uma duração prevista $d(v_i)$ e um fluxo de caixa associado $fc(v_i)$ avaliado para uma janela de oportunidade *P*, e
- E_G é um conjunto de pares ordenados , tal que se $(v_a, v_b) \in E_G$, então v_b depende de v_a.
- Se (v_a, v_b) e $(v_c, v_b) \in E_G$, então $v_a = v_c$.

No grafo apresentado na Figura 2.1, $V_G = \{ AE_1, AE_2, A, B, C\}$ e $E_G = \{ (AE_1, A), (AE_2, C), (A, B), (B, C)\}$. Uma introdução à teoria de grafos pode ser encontrada em (GROSS; YELLEN, 2005).

2.1.4 Valor presente líquido

Naturalmente, o valor de um projeto depende da ordem em que as unidades do *software* são produzidas. Por exemplo, se as unidades de *software* da Figura 2.1 forem desenvolvidas na ordem $AE_2 \rightarrow AE_1 \rightarrow A \rightarrow B \rightarrow C$, então o lucro obtido será

$$vpl(AE_2,1) + vpl(AE_1,2) + vpl(A,3) + vpl(B,4) + vpl(C,5) =$$

$$= -\$39 - \$48 + \$137 + \$130 + \$42$$

$$= \$222$$

No entanto, se a ordem de desenvolvimento for $AE_1 \rightarrow A \rightarrow B \rightarrow AE_2 \rightarrow C$, então o aumento do lucro será de

$$vpl(AE_1,1) + vpl(A,2) + vpl(B,3) + vpl(AE_2,4) + vpl(C,5) =$$

$$= -\$49 + \$169 + \$175 - \$37 + \$42$$

$$= \$300$$

Como nem toda seqüência de unidades de *software* satisfaz as relações de precedência estabelecidas (veja Figura 2.1), é importante definir formalmente o que é uma seqüência válida. Assim sendo, uma seqüência válida S é um conjunto ordenado de unidades de *software* tal que:

- Todas as unidades de *software* aparecem apenas uma vez na seqüência,
- O desenvolvimento de uma unidade de *software* só pode começar depois que todos os seus predecessores tiverem sido completamente desenvolvidos, e
- Não existe nenhum intervalo de tempo entre o fim do desenvolvimento de uma unidade e o início do desenvolvimento da próxima, com exceção da última.

Neste exemplo, existem dez seqüências válidas, que são apresentadas na Tabela 2.3.

Já que é possível que algumas unidades de *software* demorem mais de um período para serem desenvolvidas, existe uma função $d(s_i)$ que associa a cada unidade de *software* s_i o número de períodos necessários ao seu desenvolvimento. Em termos financeiros, a soma dos VPLs de uma seqüência válida de MMFs e AEs correspondente a um dado projeto G é o "Valor Presente Líquido de S", ou *vpl(S)*. Em linguagem formal

$$vpl(S) = vpl(s_1,1) + \sum_{i=2}^{|S|} vpl(s_i, 1 + \sum_{j=1}^{i-1} d(s_j))$$

#	Seqüencia
1	$AE_1 \rightarrow A \rightarrow B \rightarrow AE_2 \rightarrow C$
2	$AE_1 \rightarrow A \rightarrow AE_2 \rightarrow B \rightarrow C$
3	$AE_1 \rightarrow A \rightarrow AE_2 \rightarrow C \rightarrow B$
4	$AE_1 \rightarrow AE_2 \rightarrow A \rightarrow B \rightarrow C$
5	$AE_1 \rightarrow AE_2 \rightarrow A \rightarrow B \rightarrow C$
6	$AE_1 \rightarrow AE_2 \rightarrow C \rightarrow A \rightarrow B$
7	$AE_2 \rightarrow AE_1 \rightarrow A \rightarrow B \rightarrow C$
8	$AE_2 \rightarrow AE_1 \rightarrow A \rightarrow C \rightarrow B$
9	$AE_2 \rightarrow AE_1 \rightarrow C \rightarrow A \rightarrow B$
10	$AE_2 \rightarrow C \rightarrow AE_1 \rightarrow A \rightarrow B$

Tabela 2.3: Seqüências válidas de unidades de software.

Onde $S = s_1 \frown s_2 \frown \cdots \frown s_m$ é uma seqüência válida de unidades de *software* pertencentes a V, $|S|$ é o número de unidades de *software* na seqüência, i é o período no qual s_i é desenvolvido e $d(s_j)$ é a duração do desenvolvimento da unidade s_j.

2.1.5 Cadeias

Uma cadeia $s_1 \frown s_2 \frown \cdots \frown s_m$ é uma subseqüência de uma seqüência válida tal que s_1 é o predecessor imediato de s_2 no grafo de precedências do projeto, s_2 é o predecessor imediato de s_3 e assim por diante. Por exemplo, considerando o grafo de precedências da Figura 2.1 e a seqüência válida

$$AE_2 \rightarrow AE_1 \rightarrow A \rightarrow B \rightarrow C,$$

temos que $AE_1 \rightarrow A \rightarrow B$ e $A \rightarrow B$ são cadeias. No entanto, $AE_1 \rightarrow AE_2$ não é uma cadeia, pois AE_1 não é predecessor imediato de AE_2 no grafo.

O Valor Presente Líquido de uma cadeia C qualquer (*VPLC*), quando desenvolvida a partir do período i, ou $vplc(C,i)$, é a contribuição financeira de C para o valor de um projeto. Por exemplo, o $vplc(AE_1 \rightarrow A \rightarrow B, 2)$ é dado por

$$vpl(AE_1,2) + vpl(A,3) + vpl(B,4)$$

Em termos formais

$$vplc(C,k) = vpl(s_1,k) + \sum_{i=2}^{|C|} vpl(s_i, k + \sum_{j=1}^{i-1} d(s_j)) \tag{2.1}$$

Onde $C = s_1 \frown s_2 \frown \cdots \frown s_m$ é uma cadeia de unidades de *software* pertencentes a V, $|C|$ é o número de unidades de *software* na cadeia C e k é o período em que começa o desenvolvimento da cadeia.

2.1.6 Ponderação do VPL da cadeia

Denne e Cleland-Huang (DENNE; CLELAND-HUANG, 2004b) observaram empiricamente que a acurácia do IFM na identificação da seqüência ótima de desenvolvimento aumenta em muito se o valor de cada cadeia for negativamente ponderado de acordo com o número de períodos necessários ao seu desenvolvimento. Portanto, a contribuição de uma cadeia para o valor de um projeto é melhor expressada pelo seu VPL Ponderado, ou *VPLP*. Em termos formais:

$$vplp(C,k) = vplc(C,k) \times (1 - (FP \times (d(C) - 1))) \tag{2.2}$$

Onde C é uma cadeia, FP é um fator de ponderação escolhido e $d(C)$ é o número de períodos necessários para desenvolver C.

Em (DENNE; CLELAND-HUANG, 2004b), Denne e Cleland-Huang sugerem que o valor de *FP* para uma cadeia que leve 8 períodos para ser desenvolvida deva ser um número entre 10% e 15%, e que para uma cadeia que leve 16 períodos o fator deva estar entre 20% e 25%.

2.1.7 A heurística IFM

A seguir apresentamos os passos básicos da estratégia utilizada pelo IFM para sequenciar os MMFs e AEs.

Algoritmo

1. Extrair as cadeias habilitadas (sem predecessores pendentes) do grafo de precedências;
2. Avaliar o *VPL* de cada cadeia habilitada, considerando seus desenvolvimentos como iniciando no primeiro período livre;
3. Avaliar o *VPLP* de cada cadeia habilitada;
4. Selecionar a cadeia com o maior *VPLP*;
5. Escalar o primeiro elemento da cadeia selecionada para ser desenvolvido no primeiro período livre; e
6. Se restar algum MMF ou AE para ser escalado, voltar para o Passo 1.

2.1.8 Exemplo de execução da heurística IFM

Considere o projeto composto por unidades de *software* da Figura 2.1.

Passo 1: Extraindo as cadeias habilitadas

Neste momento, as cadeias habilitadas que podem ser extraídas do grafo de precedências apresentado na Figura 2.1 são as seguintes:

- $AE_1 \rightarrow A \rightarrow B$,
- $AE_1 \rightarrow A$,
- AE_1,
- $AE_2 \rightarrow C$ e
- AE_2.

Passo 2: Avaliando os VPLs

A Tabela 2.4 mostra os *VPLs* das cadeias habilitadas no período 1, calculadas de acordo com a Fórmula 2.1.

Cadeia	***VPL***	***VPLP***
$AE_1 \rightarrow A \rightarrow B$	295	236
$AE_1 \rightarrow A$	120	108
AE_1	-49	-49
$AE_2 \rightarrow C$	84	75
AE_2	-39	-39

Tabela 2.4: VPLs e VPLPs das cadeias habilitadas no período 1 com 10% de fator de peso.

Passo 3: Avaliando os VPLPs

Os *VPLPs* das cadeias habilitadas são avaliadas como indicado na Fórmula 2.2 e também são apresentados na Tabela 2.4.

Passo 4: Selecionando a cadeia com o maior VPLP

A cadeia que mais contribui para o valor do projeto é $AE_1 \rightarrow A \rightarrow B$, que está avaliada em $236.

Passo 5: Escalando o primeiro elemento da cadeia selecionada

O desenvolvimento da unidade AE_1 é marcado para iniciar no primeiro período livre.

Passo 6: Verificando se ainda restam unidades de software *a serem escaladas*

Como ainda existem quarto unidades de *software* para serem escaladas (AE_2, A, B e C), volte para o Passo 1 do algoritmo.

Passo n: Resultado Final

Este processo continua até que todas as cinco unidades de *software* tenham sido escaladas. Nesse momento, a seqüência encontrada será

$$AE_1 \rightarrow A \rightarrow B \rightarrow AE_2 \rightarrow C$$

que possui um *VPL* de \$300 mil reais e é o escalonamento mais lucrativo para o projeto.

2.1.9 Limitações do IFM

Para encontrar um resultado em tempo polinomial, a heurística IFM se vale das seguintes limitações:

- Cada módulo do projeto possui no máximo 1 predecessor;
- A heurística não trata situações de escassez de recursos;
- O adiamento do desenvolvimento de MMFs e AEs não é considerado, de forma que sempre hajam módulos em execução até o término do projeto;
- O IFM não garante que a solução gerada seja ótima.

O método proposto nesta dissertação supera essas limitações, fornecendo a solução ótima para projetos com estruturas de dependências complexas, considerando a possibilidade de adiamento e restrições de recursos de diversos tipos como: tempo, capital, recursos renováveis e não renováveis.

2.2 Classificação do Problema

Os esforços na área de escalonamento de projetos levam a uma grande variedade de problemas, o que motivou o uso de um esquema de classificação para descrever o propósito desse método. O esquema utilizado aqui é apresentado em (DEMEULEMEESTER; HERROELEN, 2002). Ele é baseado no esquema padrão para problemas de escalonamento de máquinas e também é composto de três áreas $\alpha|\beta|\gamma$. Nesse tipo de problema, a primeira área α serve para descrever o ambiente da máquina; a segunda área β serve para descrever as características das tarefas e dos recursos; e a terceira área γ indica o critério de otimização (indicador de desempenho).

Sendo assim, a classe de problemas que esse método atinge é descrita nas Tabelas 2.5, 2.6 e 2.7 que apresentam as classificações nos aspectos α, β e γ, respectivamente. Essa classificação nos permite descrever resumidamente o problema como: $m,1,T,k\,|\,cpm,\delta_n,c_j\,|\,nonreg,npv$.

Onde a função objetivo é:

$$max \sum_{i=2}^{n-1} c_i e^{-\alpha(s_i+d_i)} \quad ; \qquad c_i = \sum_{t=1}^{|P|-s_i+1} g_{it} e^{\alpha(|P|-s_i+1-t)}$$

Sujeito a

$$s_i + d_i \leq s_j \quad \forall (i,j) \in E_G$$

$$s_i \in \mathbb{N} \qquad \text{para } i = 2, ..., n$$

$$s_1 = 1$$

$$d_1 = 0$$

$$d_n = 0$$

$$f_i \leq \delta_n \qquad \text{para os módulos que serão desenvolvidos.}$$

$$f_1 = 1$$

$$|P| \geq \delta_n$$

$$\sum_{i \in S(t)} w_{ik} \leq a_k \qquad \text{para } k = 1, ..., m \text{ e } t = 1, ..., \delta_n$$

Onde

- A função objetivo maximiza o valor presente líquido do projeto através da escolha correta dos valores de s_i.
- i representa apenas os módulos do projeto que serão desenvolvidos.
- Os módulos do projeto estão ordenados topologicamente em $i = 1, ..., n$, de forma que as arestas de dependência sempre apontem de um número menor para um maior, onde 1 é a ordem da atividade *Start* e n é a ordem da atividade End.
- c_i representa o valor final (somatório com desconto) de todos os fluxos de caixa ocorridos a partir do desenvolvimento do módulo i.
- g_{it} representa o fluxo de caixa que ocorre para o módulo i no período $t = s_i, ..., |P|$.
- s_i representa o período de início do módulo i.
- d_i representa a duração do desenvolvimento do módulo i.
- δ_n representa o prazo limite para a conclusão do projeto.
- f_i representa o período de término do desenvolvimento do módulo i ($f_i = s_i + d_i - 1$).

- $|P|$ representa o fim do intervalo de tempo P, para o qual os valores das MMFs estão definidos.
- $e^{-\alpha} = 1/(1+r)$ é a taxa de desconto que corrige os valores do fluxo de caixa no tempo
- w_{ik} é a quantidade de recurso k necessária para o desenvolvimento do módulo i.
- a_k é a quantidade disponível do recurso k.
- E_G é o conjunto de arestas no grafo de dependências, onde cada elemento (i, j) representa uma dependência onde um módulo i deve ser concluído para que um outro módulo j possa ser desenvolvido.
- $S(t)$ é o conjunto de módulos que estarão em desenvolvimento durante o período t

Em decorrência da forma especial como o recurso financeiro é tratado, este não se enquadra em nenhum dos tipos (parâmetro α_3 da Tabela 2.5) apresentados em (DEMEULEMEESTER; HERROELEN, 2002). A principal razão é que as receitas geradas no decorrer do projeto, com a conclusão dos módulos, podem ser aplicadas no desenvolvimento dos módulos remanescentes.

Parâmetro	**Descrição do parâmetro**	**Valor**	**Descrição do valor**
α_1	Especifica a estrutura de máquinas do processo de execução.	°	*Nenhum recurso estrutural* (máquina) é especificado no problema de escalonamento.
α_2	Especifica o *número de recursos* de um problema de escalonamento de projeto (além das máquinas).	m	O número de tipos de recursos é igual à m.
α_3	Informa os *tipos específicos de recursos* utilizados.	1	*Recursos renováveis*, cuja disponibilidade é especificada para o período de duração da unidade (ex. hora, turno, dia, semana, mês, ...).
		T	*Recursos não renováveis*, cuja disponibilidade é especificada para a duração máxima T do projeto.
α_4	Descreve as características da *disponibilidade de recursos* do problema de escalonamento de projeto.	k	Os *Recursos renováveis* estão disponíveis em uma quantidade constante de no máximo k unidades ao longo do tempo.

Tabela 2.5: A classificação α do problema.

Parâmetro	Descrição do parâmetro	Valor	Descrição do valor
β_1	Sinaliza a possibilidade de preempção das atividades.	$\circ$	Nenhuma preempção é permitida.
β_2	Reflete as restrições de precedência.	cpm	Apenas precedências do tipo Fim-Início com tempo zero de intervalo (*lag*), como é usado no modelo básico PERT/CPM.
β_3	Descreve o tempo necessário à disponibilização dos pacotes.	$\circ$	O tempo de disponibilização dos pacotes (*ready time*) é sempre zero.
β_4	Descreve a duração das atividades do projeto.	$\circ$	As durações das atividades são valores inteiros arbitrários fornecidos como parâmetros.
β_5	Descreve os prazos.	δ_n	Um prazo determinístico é imposto ao projeto.
β_6	Referencia a natureza dos recursos requeridos.	$\circ$	As atividades requerem recursos em uma quantidade constante, discreta e arbitrária (ex.: número de unidades para todo o tempo de execução da atividade).
β_7	Descreve o tipo e o número de modos de execução possíveis para as atividades do projeto.	$\circ$	As atividades devem ser realizadas em um único modo de execução (uma única estimativa para custo, duração, recursos necessários, ...).
β_8	É usado para descrever as implicações financeiras das atividades do projeto.	c_j	As atividades estão associadas à um fluxo de caixa determinístico e arbitrário.
β_9	É usado para referenciar os tempos de transporte.	$\circ$	Nenhum tempo de transporte (*change-over*) é considerado.

Tabela 2.6: A classificação β do problema.

Parâmetro	Descrição do parâmetro	Valor	Descrição do valor
γ	Referencia o critério de otimização (indicador de desempenho).	*nonreg*	A medida de desempenho é qualquer medida parcial (não regular).
		npv	Maximiza o valor presente líquido do projeto.

Tabela 2.7: A classificação γ do problema.

.

2.3 Sobre o método Branch & Bound

De acordo com (LIBERTI, 2003) os algoritmos *Branch & Bound* são muito bem sucedidos e frequentemente aplicados a problemas de otimização não linear.

2.3.1 Como o Branch & Bound funciona

Dividir para conquistar é o conceito básico por trás deste método. Como o tamanho e a complexidade do problema original dificultam a solução direta, divide-se o problema sucessivamente em subproblemas cada vez menores até que apresentem tamanho e complexidade passíveis de conquista (solução).

A divisão (*branch*) é feita particionando-se o conjunto de soluções válidas em subconjuntos cada vez menores. A conquista é feita, em parte, calculando-se um limite (*bound*) de quão boa a melhor solução do subconjunto pode vir a ser. Subconjuntos são descartados quando seus limites indicam que não existe a possibilidade deles conterem uma solução ótima para o problema original.

Esta estratégia nos leva a um algoritmo composto por dois passos, que geram uma árvore de busca para encontrar a solução ótima. Enquanto o passo do *branch* é responsável por fazer com que a árvore cresça, o passo do *bound* é responsável por limitar esse crescimento (HILLIER; LIEBERMAN, 2001).

Em termos formais, a árvore de busca pode ser representada por $T(\Omega_T, \Theta_T)$ onde $\Omega_T = \{v_1, v_2, \cdots, v_n\}$ é o conjunto de vértices ou nós, $\Theta_T = \{(v_i, v_j) | v_i, v_j \in \Omega_T \wedge pai(v_j) = v_i)\}$ é o conjunto de arestas e a raiz é sempre o nó associado ao vértice *Start* do grafo de dependências. A função $pai(N)$ retorna o nó que antecede N na árvore de busca.

A decisão de podar a árvore utiliza duas funções denominadas ”limite superior”e ”limite inferior”. A função limite superior de N, ou $ls(N)$, é uma heurística otimista que estima o valor máximo da melhor solução que pode ser encontrada a partir de um nó candidato N da árvore de busca. A função limite inferior $li(N)$ faz justamente o oposto, isto é, estima o valor mínimo da melhor solução que pode ser encontrada a partir de um nó candidato N.

Um nó só pode ser descartado quando seu limite superior for menor que o maior limite inferior já encontrado pela busca. Neste caso, é impossível que ele contenha uma solução ótima, pois existe outro nó que contém uma solução melhor que qualquer outra que o nó descartado possa oferecer.

No caso do método proposto, nós intermediários também podem representar seqüências válidas. Como a limitação de recursos pode inviabilizar a construção de todas as partes do projeto, é realizado um relaxamento na definição de seqüência válida, passando a incluir nessa classe as seqüências incompletas que representam um desenvolvimento parcial do projeto.

2.3.2 Classificação do método

Segundo (DEMEULEMEESTER; HERROELEN, 2002) , geralmente é feita uma distinção entre duas estratégias de algoritmos *Branch & Bound*. Essa dicotomia se baseia na escolha do próximo nó onde será realizado o *branch*.

A primeira estratégia é a *depth-first*, ou *backtraking*, que seleciona um dos nós entre os que foram criados no estágio anterior. No entanto, se o último estágio tiver sido totalmente verificado, a busca retorna um nível acima, até encontrar um nó que não tenha sido totalmente explorado.

A segunda estratégia é tradicionalmente conhecida como *best-first*, *frontier search* ou *skiptracking*. Ela sempre seleciona o nó que apresenta o melhor valor de limite (quando apenas um limite é utilizado). Quando existem mais de um nó com o mesmo valor de limite, outros critérios também são definidos (a ordem de criação dos nós é um exemplo).

A seguir, são apresentadas as principais vantagens e desvantagens de ambas as estratégias:

Depth-First

1. Requer muito menos memória.
2. Encontra uma solução viável, e geralmente boa, quase que imediatamente e melhora esse resultado ao longo da busca.
3. Pode fornecer uma boa solução caso sua execução seja interrompida.
4. Visitas aos nós geralmente são mais rápidas.
5. Em geral, visita um número muito maior de nós.

Best-First

1. Em geral, visita um número muito menor de nós.
2. Requer muito mais memória.
3. Não fornece soluções válidas no decorrer da busca e a solução ótima só é obtida no final da busca.
4. Nenhuma solução será obtida caso sua execução seja interrompida.
5. Visitas aos nós costumam ser mais demoradas.

A estratégia que o método proposto nesta dissertação utiliza na seleção de um nó para a realização do *branch* é a *Best-First*. No entanto, as desvantagens 3 e 4 apresentadas não se aplicam, pois, devido às restrições de recursos e à forma como a busca é realizada, nós intermediários também podem representar

soluções válidas. Além disso, existem outras características que o diferenciam, como a utilização de duas heurísticas: a de limite superior e a de limite inferior, que servem como critérios para os processos de *branch* e de *bound*.

Após realizada a seleção do nó, pode-se dar continuidade à ramificação da árvore de busca (*branch*). A abordagem utilizada nessa etapa é semelhante à *Extension Alternatives* (alternativas de continuação) (STINSON EDWARD W. DAVIS, 1978; DEMEULEMEESTER; HERROELEN, 2002). Nessa abordagem, os módulos ainda não iniciados, para os quais todos os predecessores já tenham sido desenvolvidos, são combinados dando origem a conjuntos com pelo menos um módulo e que não excedem os recursos disponíveis. Cada conjunto representa uma opção de continuação do desenvolvimento do projeto no próximo nível do *branch*. Contudo, uma importante diferença do método B&B proposto para a abordagem *Extension Alternatives* de Stinson é que no método proposto o conjunto vazio também é considerado uma opção de continuação, já que o adiamento de uma despesa pode ser vantajoso.

3 *O Método Branch & Bound para Maximização do VPL*

A seguir, apresentamos os algoritmos correspondente ao método *Branch & Bound* e às funções heurísticas de limite superior e de limite inferior. Para isso, definimos também um conjunto de funções auxiliares.

3.1 Descrição do Método Proposto

Alguns métodos similares permitem que o saldo do projeto assuma valores negativos no decorrer do tempo ou consideram como válidos apenas aqueles planos que contenham todas as atividades. No entanto, o método e o exemplo apresentados nesta dissertação se referem ao caso em que o saldo do projeto deve ser sempre positivo e planos de desenvolvimento parcial também são considerados soluções válidas pelo método proposto.

Em termos formais, o algoritmo Branch & Bound que maximiza o VPL de um projeto de software é descrito conforme abaixo.

Dados:

- Um grafo de precedências entre unidades de software $G(V_G, E_G)$, composto de um conjunto de unidades de software V_G com suas respectivas previsões de fluxos de caixa, durações e demanda de recursos, e das restrições de precedência descritas em E_G;
- Uma janela de oportunidade P;
- Uma taxa de desconto r;
- Um investimento inicial C;
- Uma disponibilidade inicial de recursos renováveis e não renováveis W.

A sequência S de unidades de software $v_i \in V_G$ que apresenta o maior VPL é identificada pelo seguinte algoritmo:

$$
B\&B(G,P,r,C,W) \leftarrow \begin{cases}
\Omega_T \leftarrow \{Start\}; \\
\Theta_T \leftarrow \emptyset; \\
Q \leftarrow \{Start\}; \\
CapitalInicial \leftarrow C; \\
\textbf{Repita:} \\
\quad N \leftarrow q \in Q, \text{ such that } ls(q) = Maior(\{ls(q')|q' \in Q\}), \\
\quad Branch: \\
\quad\quad \Omega_T \leftarrow \Omega_T \cup filhos(N), \\
\quad\quad \Theta_T \leftarrow \Theta_T \cup \{(N,e)|e \in filhos(N)\}, \\
\quad Bound: \\
\quad\quad MaiorLI \leftarrow Maior(\{li(v)|v \in \Omega_T\}), \\
\quad\quad Q \leftarrow \{v \in \Omega_T | v \in (Q - \{N\}) \cup filhos(N) \\
\quad\quad\quad \wedge ls(v) \geq MaiorLI\}, \\
\quad S \leftarrow \{caminho(v)|v \in \Omega_T \wedge li(v) = MaiorLI\}, \\
\textbf{Até que: } Q = \emptyset; \\
B\&B \leftarrow S.
\end{cases}
$$

Onde

- Ω_T é o conjunto de nós na árvore de busca;
- Θ_T é o conjunto de arestas que ligam os nós da árvore de busca;
- *MaiorLI* é o maior limite inferior encontrado até o momento;
- Q é a lista de nós candidatos; e
- S é o conjunto de caminhos para as soluções ótimas que foram encontradas entre os nós de Q.

A inicialização ocorre com a inserção do nó *Start* ná árvore de busca e na lista de nós candidatos. O laço de repetição começa com a seleção do candidato que possui o maior valor de limite superior (N). No passo do *branch*, as possibilidades de continuidade do projeto a partir da situação representada pelo nó selecionado são adicionadas à árvore de busca (Ω_T e Θ_T). No passo do *bound*, a lista de nós candidatos (Q) é atualizada e os nós que certamente não conduzem à solução ótima (possuem um limite superior menor que *MaiorLI*) são descartados da lista. A melhor solução encontrada até o momento é armazenada em S. A busca só termina quando não existirem mais candidatos na lista, ou seja, quando houver certeza de que S contém a solução ótima. Caso ainda haja um ou mais candidatos na lista, o melhor deles é selecionado para a realização do próximo *branch*.

3.2 Funções Auxiliares

São funções simples, e muitas vezes já conhecidas, utilizadas na descrição do método e de outras funções mais complexas.

- *inicio*($v \in V_G, N \in \Omega_T$) $\rightarrow \mathbb{N}$ retorna o primeiro período de desenvolvimento do módulo *v*, de acordo com o caminho da árvore de busca que segue da raiz em direção a N.
- *escalonados*($N \in \Omega_T$) $\rightarrow \{V_G\}$ retorna o conjunto de módulos do projeto que estão ou já estiveram em desenvolvimento de acordo com a situação representada pelo nó N da árvore.
- *vplMaximo*($v \in V_G, N \in \Omega_T$) $\rightarrow \mathbb{R}$ retorna o maior valor que o VPL do módulo *v* pode apresentar, considerando os escalonamentos já realizados ao longo do caminho de N na árvore de busca.
- *fim*($v \in V_G, N \in \Omega_T$) $\rightarrow \mathbb{N}$ retorna o último período de desenvolvimento do módulo *v*, de acordo com o caminho de N na árvore de busca.
- *periodo*($N \in \Omega_T$) $\rightarrow \mathbb{N}$ retorna o período representado pelo nó N na árvore.
- *atividades*($N \in \Omega_T$) $\rightarrow \{V_G\}$ retorna o conjunto de módulos do projeto que estarão em desenvolvimento durante o período representado por N.
- *caminho*($N \in \Omega_T$) $\rightarrow \{V_G \times \mathbb{N}\}$ retorna o escalonamento de módulos de software que conduz o projeto à situação representada pelo nó N na árvore de busca.
- *predecessores*($v \in V_G$) $\rightarrow \{V_G\}$ retorna o conjunto de módulos que devem estar totalmente concluídos para que se possa dar início ao desenvolvimento do módulo *v*.
- *concluidos*($N \in \Omega_T$) $\rightarrow \{V_G\}$ retorna o conunto de módulos do projeto que estarão concluídos ao término do período representado pelo nó N da árvore.
- *disponivel*($w \in W, N \in \Omega_T$) $\rightarrow \mathbb{R}$ retorna a quantidade de recurso *w* que estará disponível no início do período seguinte ao representado pelo nó N. O cálculo é realizado de forma diferente para cada tipo de recurso (renovável, não renovável, tempo ou capital).
- *demanda*($w \in W$, $v \in V_G$) $\rightarrow \mathbb{R}$ retorna a quantidade de recurso *w* que é necessária para o desenvolvimento do módulo *v*.

3.3 A Função de Limite Superior

A função de limite superior *ls*($n \in \Omega_T$) $\rightarrow \mathbb{R}$ retorna uma estimativa otimista do maior VPL que se pode encontrar nos escalonamentos gerados a partir de um determinado caminho da árvore de busca, ou

seja, a seqüência que vai da raiz até o nó *n*. Para isso, ela soma o VPL dos módulos já escalonados com o maior VPL que os módulos ainda não escalonados podem apresentar.

$$ls(N) \leftarrow CapitalInicial + \sum vpl(v_j, inicio(v_j, N)) \text{ tal que } v_j \in escalonados(N) + \sum vplMaximo(v_k, N) \text{ tal que } v_k \in V_G \wedge v_k \notin escalonados(N)$$

3.4 A Função de Limite Inferior

A função de limite inferior $li(n \in \Omega_T) \rightarrow \mathbb{R}$ retorna uma estimativa pessimista do maior VPL que se pode encontrar nos escalonamentos gerados a partir de um determinado caminho da árvore de busca, ou seja, a seqüência que vai da raiz até o nó *n*. Para isso, ela soma o VPL dos módulos já concluídos com os fluxos de caixa dos módulos que ainda se encontram em desenvolvimento durante o período representado por *n*. Por se tratar do VPL de um nó da árvore de busca, a melhor solução que se pode obter a partir desse nó é melhor ou igual ao valor do próprio nó, fazendo com que esse valor seja uma estimativa pessimista para o VPL da melhor solução nesse ramo da árvore.

$$li(N) \leftarrow CapitalInicial + \sum vpl(s_j, inicio(s_j, N)) \text{ tal que } s_j \in escalonados(N) \wedge fim(s_j, N) \leq periodo(N) + \sum_{inicio(s_k,N)}^{periodo(N)} fcd(s_k) \text{ tal que } s_k \in atividades(N) \wedge fim(s_k, N) > periodo(N)$$

3.5 A Função de Branch

A função $filhos(N \in \Omega_T) \rightarrow P(V_G)$ retorna um conjunto de partes do conjunto de módulos do projeto. Cada parte (subconjunto) de V_G que contenha apenas módulos que possam estar simultaneamente em desenvolvimento no período subseqüente ao de *N* é considerada uma situação viável para o projeto. Uma situação é viável quando os recursos disponíveis são suficientes para que ela seja alcançada, ou seja, quando for uma situação na qual o projeto possa vir a se encontrar logo após a situação representada por *N*. Posteriormente, cada situação será representada na forma de um nó descendente de *N* na árvore de busca. Sendo assim, encontrar esse conjunto de situações viáveis é equivalente a encontrar:

$$S_1 \longleftarrow \{v \in V_G \text{ tal que } (v, inicio(v,N)) \notin caminho(N) \wedge predecessores(v) \subseteq concluidos(N))\}$$

$$S_2 \longleftarrow \{v \in S_1 \text{ tal que } \forall w(disponivel(w,N) \geq demanda(w,v))\}$$

$$S_3 \longleftarrow \{v \text{ tal que } v \in atividades(N) \wedge fim(v,N) > periodo(N)\}$$

$$S_4 \longleftarrow \{S_3 \bigcup s \text{ tal que } s \in P(S_2)\}$$

$$filhos(N \in \Omega_T) \longleftarrow \{s \in S_4 \text{ tal que } \forall w(disponivel(w,N) \geq \sum_{v \in s} demanda(w,v))\}$$

Onde

- S_1 é o conjunto de módulos sem predecessores pendentes;
- S_2 é o conjunto de módulos sem predecessores pendentes e cujos requisitos podem ser satisfeitos pela disponibilidade atual de recursos.
- S_3 é o conjunto de módulos que se encontram em desenvolvimento no período representado por N e que continuarão em desenvolvimento no período subseqüente.
- S_4 é o conjunto das uniões de S_3 com as partes de S_2. Cada elemento de S_4 representa uma potencial situação posterior à N, dependendo apenas de uma verificação na disponibilidade de recursos para ser considerada uma situação viável.

4 *Exemplo de Aplicação*

Nesta seção, apresentamos um exemplo prático de aplicação do método proposto. A execução é realizada passo-a-passo em um projeto inspirado no mundo real sobre o desenvolvimento de um software para dispositivos móveis, tais como: celulares, *smartphones* e *palm tops*.

4.1 Contexto do Projeto

Empréstimos consignados são empréstimos genéricos de baixo risco e juros reduzidos, concedidos a empregados seletos de empresas sólidas. Nesses empréstimos o pagamento é realizado em parcelas, por meio de dedução direta nos salários. Sendo assim, considere uma instituição financeira internacional como o CITIBANK, BARCLAYS, HSBC, ABN AMRO, UBS e muitas outras que disponibilizam empréstimos consignados para seus clientes. Para o propósito deste exemplo, essa organização se chamará LOANS "R" US, ou simplesmente *LRU*.

Assim que recebe uma requisição de empréstimo, a *LRU* estima as chances desse cliente conseguir pagar o empréstimo de acordo com valores e datas de parcelamento aceitáveis. Em seguida, a *LRU* verifica se possui fundos suficientes para realizar o empréstimo requisitado, já que, por lei, as instituições financeiras não podem emprestar dinheiro além de um determinado limite, de forma a preservar sua saúde financeira. Por fim, as condições para a realização do empréstimo consignado são apresentadas ao cliente, caso seja possível realizá-lo. Nesse momento, o cliente tem a opção de aceitar ou recusar a oferta de empréstimo.

Instituições financeiras que realizam empréstimos geralmente oferecem serviços de refinanciamento de dívidas como forma de fortalecer o vínculo com seus clientes atuais e futuros. A *LRU* não é uma exceção à esta regra. Como a operação de refinanciamento pode envolver um contrato de empréstimo realizado com terceiros, esta operação é normalmente chamada de *compra de dívidas*.

Para responder adequadamente aos recentes movimentos da concorrência no mercado de empréstimos consignados e melhorar a qualidade dos serviços que presta a seus clientes, a *LRU* pretende investir em um novo sistema de empréstimos via web para dispositivos móveis. A companhia acredita que, se agir rápido, este software pode não apenas aumentar consideravelmente seu faturamento, mas também redefinir favoravelmente o competitivo mercado dos empréstimos consignados.

A Figura 4.1 introduz o diagrama de precedência dos módulos do novo software de controle de empréstimos, enquanto a Tabela 4.1 descreve o significado e o tipo de cada módulo apresentado na figura.

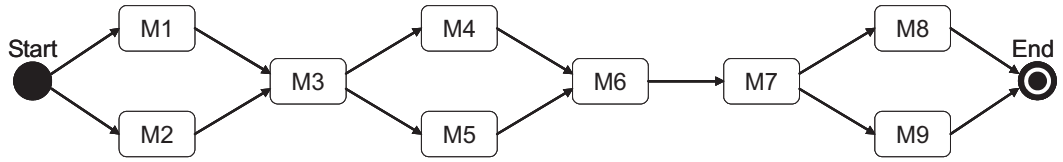

Figura 4.1: Os módulos do projeto e suas dependências.

Módulos do Software			
Id	**Nome**	**Descrição**	**Tipo**
M1	Solicitar empréstimo	Coleta as informações necessárias para a realização de um empréstimo.	MMF
M2	Solicitar refinanciamento	Coleta as informações necessárias para o refinanciamento de um empréstimo existente.	MMF
M3	Análise rápida de crédito	Estima as chances de um cliente conseguir pagar um empréstimo de acordo com as datas e valores do parcelamento.	MMF
M4	Análise minuciosa de crédito	Realiza uma análise de crédito mais detalhada sobre as requisições rejeitadas pela "Análise rápida de crédito", procurando conceder crédito às solicitações que se mostrem viáveis.	MMF
M5	Contratar seguro de vida	Para conceder certos empréstimos, a *LRU* inclui o custo do seguro de vida em pedidos de empréstimo que foram rejeitados pela "Análise rápida de crédito" devido à curta expectativa de vida dos clientes.	AE
M6	Verificar a disponibilidade de fundos	Verifica se a *LRU* possui o capital necessário para a realização de um empréstimo.	AE
M7	Gerenciar empréstimos	Determina a data mais próxima em que a *LRU* poderá satisfazer um pedido de empréstimo ou refinanciamento que tenha sido rejeitado pelo "Verificar a disponibilidade de fundos" devido à uma falta temporária de capital.	AE
M8	Aceitar condições de empréstimo	Permite que o cliente aceite ou rejeite as condições de empréstimo propostas pela *LRU*.	MMF
M9	Aceitar condições de refinanciamento	Permite que o cliente aceite ou rejeite as condições de refinanciamento propostas pela *LRU*.	MMF

Tabela 4.1: Lista de módulos do software a ser desenvolvido.

4.2 Determinando como Os Módulos Podem Gerar Valor

Devemos observar que seis dos nove módulos listados na Tabela 4.1 são unidades auto-contidas de software que agregam valor ao negócio. Estas unidades geram valor para a *LRU* da seguinte forma:

- *M1 e M2* - Todo cliente que solicita um empréstimo em folha é questionado quanto à possibilidade de receber ou não, periodicamente, novas ofertas da *LRU* e empresas associadas. Clientes em potencial podem ser alvo de novas campanhas de marketing, gerando oportunidades de venda para a *LRU* e ganhos na forma de taxas cobradas das empresas associadas pelo uso das informações contidas na base de dados da *LRU*.

- *M3* - O resultado da *Análise Rápida de Crédito* é utilizado para enriquecer a base de dados de clientes da *LRU*. Essa informação é particularmente importante para as campanhas de marketing, que dependam da situação financeira dos clientes potenciais, especialmente as campanhas que permitem aos clientes realizarem o pagamento por produtos e serviços de forma parcelada. As empresas associadas à *LRU* devem pagar uma taxa *premium* para terem acesso à base de dados enriquecida com essas informações.

- *M4* - A *Análise Minuciosa de Crédito* enriquece ainda mais a base de dados de clientes com informações que apontam oportunidades para novas campanhas de marketing da *LRU* e das empresas associadas. O retorno proporcionado por esse módulo é oriundo da taxa *premium* paga pelas empresas associadas para acessar a base de dados enriquecida dos clientes, além de proporcionar um maior número de empréstimos e refinanciamentos.

- *M8 e M9* - Uma vez que um cliente aceite uma oferta de empréstimo ou refinanciamento, estas unidades do projeto geram valor na forma de juros pagos pelos clientes.

Portanto, M1, M2, M3, M4, M8 e M9 são pacotes mínimos comercializáveis, ou MMFs (*minimum marketable features*). Por outro lado, M5, M6 e M7 são elementos arquiteturais, pois fornecem serviços necessários ao funcionamento do software, mas nem os clientes, nem as empresas associadas, estão dispostos a pagar por eles.

4.3 Restrições ao Desenvolvimento do Projeto

Os recursos de que a *LRU* dispõe para investimentos em tecnologia são limitados. Para garantir que esses recursos serão aplicados da melhor forma possível, a *LRU* espera que seus gerentes tomem as ações necessárias para garantir o maior retorno possível dos valores investidos em seus projetos de software.

Para calcular a expectativa de retorno de um projeto é necessário avaliar os recursos necessários ao desenvolvimento de cada módulo e o retorno que se espera obter deles no decorrer de um determinado

intervalo de tempo. No caso deste projeto, os módulos deverão ser avaliados ao longo de um intervalo de 15 meses. A Tabela 4.2 exibe parte das informações necessárias à essa avaliação. Para cada módulo, é exibido o seu fluxo de caixa, que é a sucessão de variações esperadas no fluxo de caixa da *LRU* a partir do seu desenvolvimento.

Módulo do Projeto	**Fluxos de Caixa por Período**					
	1	2	3	4	5 → 14	15
M1	-50	50	70	100	100	50
M2	-50	-50	20	28	40	20
M3	-80	90	120	180	180	90
M4	-90	-90	90	100	130	90
M5	-140	0	0	0	0	0
M6	-10	0	0	0	0	0
M7	-10	0	0	0	0	0
M8	-15	-15	725	1.015	2.175	1.450
M9	-40	290	410	870	870	290

Tabela 4.2: O fluxo de caixa dos módulos do projeto com valores em milhares de reais.

Com base nos fluxos de caixa apresentados, é possível determinar o VPL de cada módulo em função do mês em que será dado início ao seu desenvolvimento. Para isso, deve-se considerar uma taxa de desconto. Neste projeto, optou-se por utilizar uma taxa de desconto mensal de 2% por período, ou 2% pp, que é a taxa de custo de capital da *LRU*, isto é, a taxa de juros que a LRU paga para obter empréstimos no mercado. O resultado dessa avaliação é apresentado na Tabela 4.3.

Períodos	**Módulos do Projeto**								
	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
1	1.024	294	1.848	1.148	-137	-10	-10	20.718	8.865
2	968	274	1.746	1.060	-135	-10	-10	19.256	8.480
3	876	240	1.581	945	-132	-9	-9	17.294	7.680
4	786	206	1.419	831	-129	-9	-9	15.371	6.895
5	698	173	1.260	720	-127	-9	-9	13.485	6.126
6	611	140	1.104	612	-124	-9	-9	11.636	5.373
7	526	108	951	505	-122	-9	-9	9.823	4.633
8	443	77	801	400	-119	-9	-9	8.046	3.909
9	362	46	655	298	-117	-8	-8	6.304	3.198
10	282	16	511	197	-115	-8	-8	4.596	2.502
11	203	-13	370	99	-113	-8	-8	2.922	1.819
12	127	-42	231	2	-110	-8	-8	1.280	1.150

Tabela 4.3: O valor presente líquido dos módulos do projeto em milhares de reais.

A *LRU* procura manter-se sempre atualizada quanto à movimentação da concorrência e já identificou que uma de suas rivais pretende lançar um serviço similar em aproximadamente um ano. Sendo assim, uma das limitações impostas pela *LRU* foi o prazo de 12 meses para a conclusão do projeto. Além

da questão do tempo, existem outras limitações que restringem a forma como esse projeto deve ser conduzido. Inicialmente, a companhia pretende disponibilizar um capital R$ 100.000,00 e contratar uma equipe terceirizada de 4 pessoas para o projeto, composta de um gerente não exclusivo, que será o líder do projeto, e três programadores. A empresa contratada possui custos diferenciados que dependem do tamanho da equipe, e uma equipe com mais de 3 programadores possui custos proibitivos para a situação atual da *LRU*. O custo da equipe é calculado com base nas horas trabalhadas em cada módulo do projeto e, por isso, o custo de uma possível ociosidade desses funcionários será de responsabilidade da contratada, que poderá alocá-los em outros projetos durante esses intervalos. A Tabela 4.4 mostra, os recursos necessários ao desenvolvimento de cada módulo. Neste caso, temos que os módulos M1, M2, M3, M4, M6, M7 e M8 necessitarão do trabalho de apenas 1 programador cada, mas os módulos M5 e M9 necessitarão de 2 programadores cada. Também podemos observar que os módulos com duração maior que um período possuem o valor de custo total diferente do custo inicial. O motivo é que o custo inicial corresponde ao custo estimado apenas para o primeiro período de desenvolvimento.

Módulo do Projeto	**Duração Prevista**	**Custo Inicial**	**Custo Total**	**Programadores Necessários**
M1	1	-50	-50	1
M2	2	-50	-100	1
M3	1	-80	-80	1
M4	2	-90	-180	1
M5	1	-140	-140	2
M6	1	-10	-10	1
M7	1	-10	-10	1
M8	2	-15	-30	1
M9	1	-40	-40	2

Tabela 4.4: Recursos necessários ao desenvolvimento dos módulos do projeto.

4.4 Planejando o Desenvolvimento do Software

O líder, sendo o responsável pelo sucesso do projeto, decidiu elaborar um plano de desenvolvimento que maximize o valor do projeto para a empresa, ou seja, que ofereça o VPL máximo que o projeto pode ter. Para isso, ele optou por utilizar o método *Branch & Bound* que permite encontrar a melhor solução possível dentro das condições apresentadas.

Passo 0: Inicializando a busca

A busca é inicializada com a criação do primeiro nó da árvore de busca, que é o nó de número 0 (zero). Esse nó simboliza o início (*Start*) do prazo de 12 meses para a conclusão do projeto. Nesse ponto, já é possível estimar um intervalo de valores no qual o VPL do melhor plano de desenvolvimento possível, que ainda será encontrado, pode estar. O valor que marca o início desse intervalo é o *limite*

inferior do VPL do plano de desenvolvimento e o valor que marca o fim é o seu *limite superior*.

Como o capital disponível para o projeto é de $100 mil, pode-se assumir que o pior plano seria não desenvolver nenhum módulo. Assim, os 15 meses terminaríam com o mesmo valor de agora e o VPL da não realização do projeto seria de 100 mil reais. Esse valor é o Limite Inferior do nó 0, ou $li(0)$.

$$li(0) \leftarrow 100$$

É importante registrar esse valor como sendo o *maior limite inferior* encontrado até o momento.

$$MaiorLI \leftarrow 100$$

Para o cálculo do limite superior do VPL do projeto podemos considerar o plano de desenvolvimento ideal, que talvez não seja viável, de construir todos os módulos do projeto o mais rápido possível. Para isso, devemos descobrir o período mais cedo em que cada módulo pode ser desenvolvido. Analisando as dependências apresentadas na Figura 4.1, as durações apresentadas na Tabela 4.4 e os valores do VPL de cada módulo apresentados na Tabela 4.3, podemos concluir que:

- **M1** pode ser desenvolvido a partir do 1º mês e, portanto, o maior VPL que pode apresentar é $1.024.
- **M2** também pode ser desenvolvido a partir do 1º mês e o maior VPL que pode apresentar é $294.
- **M3** depende de M1 e M2 e, por isso, não pode ser desenvolvido antes do 3º mês já que M2 tem duração prevista de dois meses. Portanto, o VPL de M3 não pode ser superior a $1.581.
- **M4** depende de M3 e, conseqüentemente, não pode ser desenvolvido antes do 4º mês, o que restringe seu VPL máximo a $831.
- **M5** também depende de M3 e não pode ser desenvolvido antes do 4º mês, mas, por se tratar de um elemento arquitetural, seu fluxo de caixa é constituído apenas de custos e isso faz com que seu VPL seja negativo e crescente ao longo do tempo. Como conseqüência, o VPL máximo que M5 pode apresentar é $0 (zero), que corresponde ao não desenvolvimento do módulo.
- **M6** depende de M4 e M5 e, portanto, não pode ser desenvolvido antes do 6º mês. Mas, assim como M5, M6 é um elemento arquitetural e isso nos leva a um VPL máximo de $0 (zero) relativo ao não desenvolvimento deste módulo.
- **M7** depende de M6 e não poderá ser desenvolvido antes do 7º mês, mas, semelhante a M5 e a M6, representa um fluxo negativo de capital e, analogamente, seu maior VPL é zero.
- **M8** depende de M7 e não pode ser desenvolvido antes do 8º mês, ficando com uma expectativa máxima de $8.046 para o seu VPL.

- **M9** também depende de M7 e não pode ser desenvolvido antes do 8º mês, podendo apresentar um VPL máximo de $3.909.

Somando o capital inicial com o VPL máximo de cada módulo, é obtido o Limite Superior do nó #0 ou $ls(0)$.

$$ls(0) \leftarrow 100 + vpl(M1,1) + vpl(M2,1) + vpl(M3,3) + vpl(M4,4) + 0 + 0 + 0 + vpl(M8,8) + vpl(M9,8) =$$

$$100 + 1.024 + 294 + 1.581 + 831 + 8.046 + 3.909 = 15.786$$

Ou seja, mesmo sem saber qual é o plano que conduzirá o projeto ao seu VPL máximo, podemos afirmar que o valor dessa solução será maior ou igual a R$ 100.000,00 e menor ou igual a R$ 15.786.427,50 (valor sem arredondamento). Caso um dos módulos não pudesse ser concluído no intevalo de 12 meses, seu VPL não seria considerado nesse cálculo.

A Figura 4.2 apresenta a descrição do nó 0 (zero). Ela inicia com o número do nó seguido do período (mês) que ele representa e das atividades escaladas para o período. Logo após, são exibidos os valores de Limite Inferior e de Limite Superior relativos aos planos que se pode obter a partir desse nó. Na quarta parte, temos um relato dos recursos que estarão disponíveis no próximo período para o desenvolvimento dos módulos. Em alguns casos, a descrição pode terminar com um marcador que indica se este nó sofreu alguma operação de *branch* ou de *bound*.

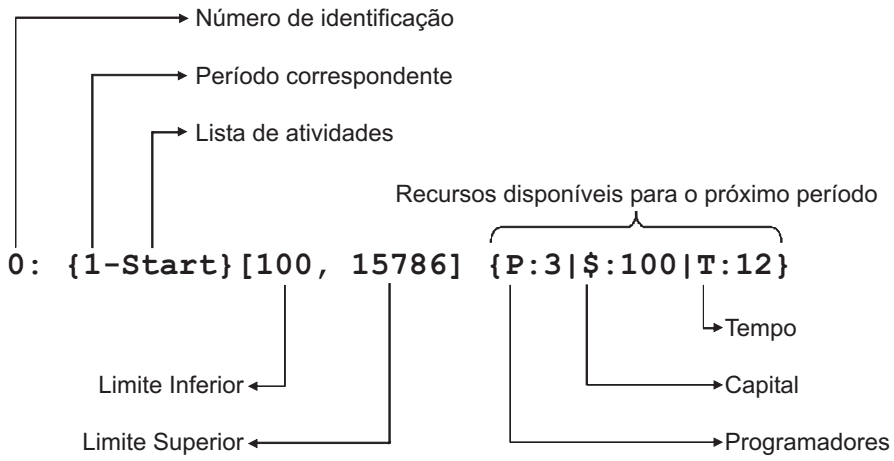

Figura 4.2: Descrição de um nó da árvore de busca.

Além de ser inserido na árvore de busca, o nó de número zero (ou nó *Start*) também precisa ser inserido na lista de nós candidatos. Essa lista deve sempre conter os nós ativos da busca, ou seja, aqueles que podem conduzir a uma solução ótima mas ainda não passaram pelo processo de *branch*.

$$Candidatos \leftarrow \{0\}$$

Passo 1: Selecionando o melhor candidato para o *branch* A busca começa com a seleção do candidato mais promissor entre aqueles que constam na lista de nós candidatos. No momento, o único candidato presente na lista é o nó 0 (*Start*), que será o escolhido.

Passo 2: Realizando o *branch*

Para abrir o leque de possibilidades é preciso identificar os módulos que poderão ser desenvolvidos no período seguinte ao do candidato 0 selecionado, ou seja, aqueles que não dependam de nenhum outro para que se inicie o seu desenvolvimento. Analisando o grafo de dependências apresentado na Figura 4.1, observa-se que os módulos M1 e M2 não possuem predecessores pendentes e, portanto, podem ser iniciados. Isso significa que existem 4 opções para o primeiro dos 12 meses de desenvolvimento:

1. **Não desenvolver nada e esperar até o próximo período** - Esta opção requer apenas a utilização de 1 mês do prazo para a conclusão do projeto.
2. **Desenvolver apenas M1** - Esta opção requer a alocação de 1 programador, 1 mês de trabalho e a utilização de $49 mil correspondente ao custo de desenvolvimento de M1 corrigido pela taxa de desconto de 2% a.m..
3. **Desenvolver M1 e M2** - Esta opção requer a alocação de 2 programadores (1 para M1 e 1 para M2), 1 mês de trabalho e a utilização de $98 mil, correspondente aos custos iniciais de M1 e M2 corrigidos pela taxa de desconto de 2% a.m. ($49 para M1 e $49 para M2).
4. **Desenvolver apenas M2** - Esta opção requer a alocação de 1 programador, 1 mês de trabalho e a utilização de $49 mil correspondente ao custo inicial de M2 corrigidos a taxa de 2% a.m..

Considerando que a disponibilidade inicial de recursos é de $100 mil reais, 3 programadores e 12 meses, e que nenhuma das opções ultrapassa esses limites, temos que todas elas são opções viáveis e, por isso, devem ser inseridas na árvore de busca como opções de continuação do nó 0.

Passo 3: Calculando os limites

No caso do nó 1, o Limite Inferior deve ser o mesmo de 0, pois a possibilidade de não desenvolver nenhuma parte do projeto foi preservada, mas o Limite Superior é menor, pois todos os módulos terão que aguardar mais 1 mês para serem desenvolvidos e, com isso, perderão 1 período de rendimentos na janela de 15 meses do cálculo do VPL.

$li(1) \leftarrow 100$

$$ls(1) \leftarrow 100 + vpl(M1,2) + vpl(M2,2) + vpl(M3,4) + vpl(M4,5) + vpl(M8,9) + vpl(M9,9)$$
$$= 100 + 968 + 274 + 1.419 + 720 + 6304 + 3198 = 12.984$$

No caso do nó 2, o Limite Inferior deve aumentar, pois M1 tem duração de 1 mês e, quando um módulo está para ser concluído antes do início do próximo período, o seu VPL deve ser considerado; mas o Limite Superior deve diminuir, pois M2 não será desenvolvido no primeiro período, impossibilitando a propagação da previsão otimista realizada no cálculo do Limite Superior do nó 0.

$$li(2) \leftarrow 100 + vpl(M1,1) = 100 + 1.024 = 1.124$$

$$ls(2) \leftarrow 100 + vpl(M1,1) + vpl(M2,2) + vpl(M3,4) + vpl(M4,5) + vpl(M8,9) + vpl(M9,9)$$
$$= 100 + 1.024 + 274 + 1.419 + 720 + 6304 + 3198 = 13.040$$

Para o nó 3, o Limite Inferior também deve aumentar, mas não tanto quanto no nó 2. O motivo é que além do VPL de M1, devemos adicionar o custo do primeiro mês de desenvolvimento de M2. Em nenhuma hipótese podemos adicionar o VPL de M2, pois esse módulo só poderá ser concluído no próximo período e não há garantias de que isso irá ocorrer, considerando que sempre poderá haver falta de recursos. O Limite superior deve continuar o mesmo de 0, pois a expectativa otimista foi preservada.

$$li(3) \leftarrow 100 + vpl(M1,1) + fcd(M2,1) = 100 + 1.024 - 49 = 1.075$$

$$ls(3) \leftarrow 100 + vpl(M1,1) + vpl(M2,1) + vpl(M3,3) + vpl(M4,4) + vpl(M8,8) + vpl(M9,8)$$
$$= 100 + 1.024 + 294 + 1.581 + 831 + 8.046 + 3.909 = 15.786$$

Para o nó 4, o Limite Inferior deve diminuir pois, assim como no caso do nó 3, temos a saída de caixa devido ao custo do primeiro mês de desenvolvimento de M2 e não há garantias de que M2 será concluído, considerando que poderá haver falta de recursos. O Limite Superior deve diminuir, pois M1 não será desenvolvido no primeiro período, impossibilitando a propagação da previsão otimista realizada no cálculo do Limite Superior do nó 0.

$$li(4) \leftarrow 100 + fcd(M2,1) = 100 - 49 = 51$$

$$ls(4) \leftarrow 100 + vpl(M2,1) + vpl(M1,2) + vpl(M3,3) + vpl(M4,4) + vpl(M8,8) + vpl(M9,8)$$
$$= 100 + 294 + 968 + 1.581 + 831 + 8.046 + 3.909 = 15.730$$

Passo 4: Calculando os recursos

Com base na disponibilidade inicial de recursos e nos gastos verificados no Passo 2, é possível estimar qual será a disponibilidade de recursos no próximo mês. Essa informação é extremamente importante para se saber quais atividades, ou combinação de atividades, poderão ser executadas no próximo mês, ajudando a limitar o crescimento da árvore de busca ao universo

das soluções viáveis.

Como a disponibilidade inicial de recursos é de 3 programadores, 100 mil reais e 12 meses de prazo, os novos cenários que surgirão no próximo mês são os seguintes:

No caso do nó 1, o único recurso consumido foi o "Tempo", já que no nó 1 não se considera o desenvolvimento de nenhum módulo e que, sendo os funcionários terceirizados, o custo dos programadores para o projeto é calculado sobre as horas trabalhadas nos módulos. Com isso, o adiamento das atividades não gera custo de pessoal. Portanto, estarão disponíveis para o próximo período: 3 programadores, 100 mil reais e 11 meses de prazo.

No caso do nó 2, que alocou 1 programador e empregou $49 mil em 1 mês de trabalho, estarão disponíveis para o próximo período: 3 programadores, 99 mil reais e 11 meses de prazo. Isso porque, como M1 tem duração de 1 mês, o programador alocado estará novamente disponível e M1 irá gerar um lucro de $50 mil, que quando corrigido pela taxa de desconto de 2% a.m. em 2 meses vale $48 mil, totalizando:

$$100 - 49 + 48 = 99 \text{ mil reais}$$

No caso do nó 3, que alocou 2 programadores e empregou $98 mil em 1 mês de trabalho ($49 mil para M1 e $49 mil para M2), estarão disponíveis para o próximo período: 2 programadores, 2 mil reais e 11 meses de prazo. Apesar de M1 ter duração de 1 mês e o programador alocado estar novamente disponível no próximo período, M2 tem duração de 2 períodos e, por isso, reterá o seu programador. Além disso, embora M1 vá gerar um lucro de $50 mil, que quando corrigidos pela taxa de desconto de 2% a.m. em 2 meses vale $48 mil, M2 irá gerar uma despesa no mesmo valor, totalizando:

$$100 - 49 - 49 + 48 - 48 = 2 \text{ mil reais}$$

No caso do nó 4, que alocou 1 programador e empregou $49 mil em 1 mês de trabalho, estarão disponíveis para o próximo período: 2 programadores, 3 mil reais e 11 meses de prazo. Isso porque M2 tem duração de 2 períodos e reterá o seu programador, além de gerar uma despesa de $50 mil no próximo período, que quando corrigido pela taxa de desconto de 2% a.m. em 2 meses vale $48 mil, totalizando:

$$100 - 49 - 48 = 3 \text{ mil reais}$$

Sendo assim, a árvore de busca atualizada passa a ter a seguinte forma:

```
0:  {1-Start}[100, 15786] {P:3|$:100|T:12} <Branched> *
.     1:  {1}[100, 12984] {P:3|$:100|T:11}
.     2:  {1-M1}[1124, 13040] {P:3|$:99|T:11}
.     3:  {1-M1,M2}[1075, 15786] {P:2|$:2|T:11}
.     4:  {1-M2}[51, 15730] {P:2|$:3|T:11}
```

Passo 5: Selecionando a melhor solução

Após inserir todos os filhos na árvore, é preciso atualizar o registro do Maior Limite Inferior. No momento, o maior valor é 1.124, relativo ao nó 2. Portanto:

$$MaiorLI \leftarrow 1.124$$

A melhor solução encontrada até momento é aquela que apresenta um Limite Inferior igual a $MaiorLI$.

Passo 6: Atualizando a lista de candidatos

Acrescente os nós obtidos no *branch* à lista de nós candidatos.

$$Candidatos \leftarrow \{0,1,2,3,4\}$$

O nó que foi expandido no *branch* deve ser removido da lista de candidatos, pois a solução que ele representa já foi avaliada e as soluções as quais ele conduz já estão sendo consideradas através de seus filhos na árvore.

Os nós que possuírem um Limite Superior menor que $MaiorLI$ também devem ser removidos da lista, pois já existe pelo menos uma solução melhor do que qualquer outra representada por eles. À essa remoção de candidatos que ainda não foram expandidos dá-se o nome de *bound*. No entanto, nesse momento nenhum nó precisou ser removido por esse motivo. Com isso, a nova lista de candidatos passou a ser:

$$Candidatos \leftarrow \{1,2,3,4\}$$

Passo 7: Verificando o critério de parada

A busca termina quando não houverem mais candidatos na lista, ou seja, quando não houver mais nenhuma possibilidade de se encontrar uma solução melhor do que aquelas que já foram encontradas.

Nesse exemplo, foram necessárias 15 ramificações (*branches*) até que esse critério fosse alcançado.

Passo 8: Selecionando o melhor candidato para o *branch*

Agora, o candidato mais promissor da lista é o nó 3, que possui um Limite Superior de $15.786. Ao realizar o *branch* nesse nó, a árvore de busca passa a ser:

```
0:  {1-Start}[100, 15786] {P:3|$:100|T:12} <Branched>
.     1:  {1}[100, 12984] {P:3|$:100|T:11}
.     2:  {1-M1}[1124, 13040] {P:3|$:99|T:11}
.     3:  {1-M1,M2}[1075, 15786] {P:2|$:2|T:11} <Branched> *
.     .     5:  {2-M2}[1419, 15786] {P:3|$:87|T:10}
.     4:  {1-M2}[51, 15730] {P:2|$:3|T:11}


MaiorLI = 1.419
Candidatos = {1, 2, 4, 5}
```

Passo n: Extraindo o plano do nó que representa a solução ótima

Como não existem mais candidatos a serem explorados, a busca termina e o plano de desenvolvimento que maximiza o VPL do projeto é o escalonamento de atividades representado ao longo do caminho que vai da raiz da árvore de busca (*Start*) até o nó cujo Limite Inferior é igual a *MaiorLI* e o VPL do projeto orientado por este plano é dado pelo próprio *MaiorLI*.

No exemplo, ao final da busca, *MaiorLI* vale $15.642 mil. O único nó que possui um Limite Inferior com esse valor é o nó 29. Portanto, o plano que apresentou o VPL máximo para o projeto, respeitando as restrições de tempo e recursos apresentadas é indicado pelo caminho $0 \frown 3 \frown 5 \frown 7 \frown 9 \frown 13 \frown 16 \frown 21 \frown 26 \frown 28 \frown 29$, e esse plano é:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M8, M9) \rightarrow (M8)$$

Onde M1 e M2 são desenvolvidos paralelamente no primeiro mês, apenas M2 é desenvolvido no segundo mês, M3 é desenvolvido no terceiro mês, M4 é desenvolvido nos meses quarto e quinto, M5 é desenvolvido apenas no quinto mês, M6 no sexto mês, M7 no sétimo, M8 é desenvolvido paralelamente a M9 no oitavo mês e individualmente no nono.

Com isso, o VPL da solução ótima, indicado por *MaiorLI*, é de R$ 15.642.039,87 (sem arredondamentos) e a conclusão do projeto se dará em 9 meses.

A árvore completa da busca é:

```
0:  {1-Start}[100, 15786] {P:3|$:100|T:12} <Branched>
.  1:  {1}[100, 12984] {P:3|$:100|T:11} <BOUNDED>
.  2:  {1-M1}[1124, 13040] {P:3|$:99|T:11} <BOUNDED>
.  3:  {1-M1,M2}[1075, 15786] {P:2|$:2|T:11} <Branched>
.  .  5:  {2-M2}[1419, 15786] {P:3|$:87|T:10} <Branched>
.  .  .  6:  {3}[1419, 13061] {P:3|$:205|T:9} <BOUNDED>
.  .  .  7:  {3-M3}[3000, 15786] {P:3|$:213|T:9} <Branched>
.  .  .  .  8:  {4}[3000, 13223] {P:3|$:448|T:8} <BOUNDED>
.  .  .  .  9:  {4-M4}[2917, 15786] {P:2|$:284|T:8} <Branched>
.  .  .  .  .  12:  {5-M4}[3831, 13334] {P:3|$:648|T:7} <BOUNDED>
.  .  .  .  .  13:  {5-M4,M5}[3704, 15660] {P:3|$:521|T:7} <Branched>
.  .  .  .  .  .  15:  {6}[3704, 13207] {P:3|$:887|T:6} <BOUNDED>
.  .  .  .  .  .  16:  {6-M6}[3695, 15651] {P:3|$:878|T:6} <Branched>
.  .  .  .  .  .  .  20:  {7}[3695, 13198] {P:3|$:1262|T:5} <BOUNDED>
.  .  .  .  .  .  .  21:  {7-M7}[3687, 15642] {P:3|$:1253|T:5} <Branched>
.  .  .  .  .  .  .  .  24:  {8}[3687, 13190] {P:3|$:1630|T:4} <BOUNDED>
.  .  .  .  .  .  .  .  25:  {8-M8}[3674, 14932] {P:2|$:1604|T:4} <BOUNDED>
.  .  .  .  .  .  .  .  26:  {8-M8,M9}[7583, 15642] {P:2|$:1813|T:4} <Branched>
.  .  .  .  .  .  .  .  .  28:  {9-M8}[15642, 15642] {P:3|$:3113|T:3} <Branched>
.  .  .  .  .  .  .  .  .  .  29:  {10-End}[15642, 15642] {P:3|$:3113|T:3} <Branched> *
.  .  .  .  .  .  .  .  27:  {8-M9}[7596, 13900] {P:3|$:1838|T:4} <BOUNDED>
.  .  .  .  10:  {4-M4,M5}[2787, 15657] {P:2|$:154|T:8} <Branched>
.  .  .  .  .  17:  {5-M4}[3702, 15657] {P:3|$:518|T:7} <Branched>
.  .  .  .  .  .  18:  {6}[3702, 13205] {P:3|$:884|T:6} <BOUNDED>
.  .  .  .  .  .  19:  {6-M6}[3693, 15648] {P:3|$:875|T:6} <Branched>
.  .  .  .  .  .  .  22:  {7}[3693, 13196] {P:3|$:1259|T:5} <BOUNDED>
.  .  .  .  .  .  .  23:  {7-M7}[3684, 15640] {P:3|$:1250|T:5} <BOUNDED>
.  .  .  .  11:  {4-M5}[2870, 13094] {P:3|$:319|T:8} <BOUNDED>
.  4:  {1-M2}[51, 15730] {P:2|$:3|T:11} <Branched>
.  .  14:  {2-M2}[394, 12912] {P:3|$:22|T:10} <BOUNDED>
MaiorLI = 15.642
Candidatos = {}
```

Note que 15 nós estão marcados como *branched*, indicando que foram expandidos, enquanto os outros 15 estão marcados como *bounded*, indicando que foram descartados no decorrer da busca.

4.5 Progressão da Busca

A Figura 4.3 mostra a progressão do valor de *MaiorLI* a cada *branch* realizado ao longo da busca, enquanto a Figura 4.4 mostra a progressão do número de nós gerados, tamanho da lista de candidatos e quantidade de nós cortados pelo *bound* a cada *branch*. A curva do número de nós gerados é crescente, mas o número de nós candidatos diminui a medida em que o número de nós cortados aumenta, e esse aumento ocorre a medida em que *MaiorLI* assume valores mais elevados.

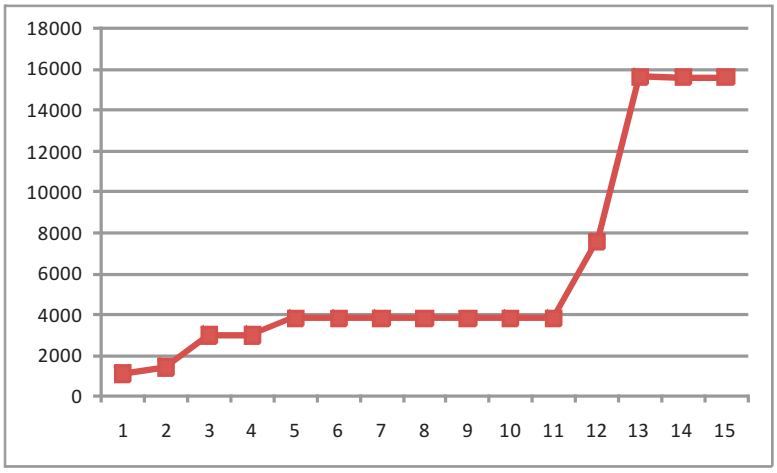

Figura 4.3: Progressão do valor de *MaiorLI* ao longo da busca.

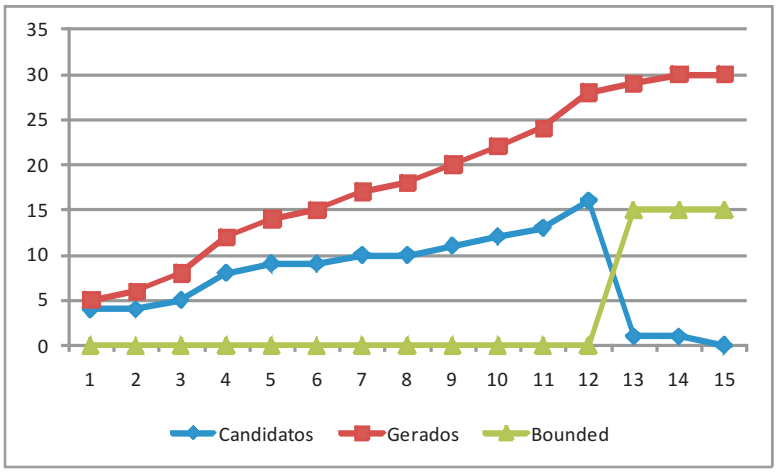

Figura 4.4: Progressão do número de nós gerados, nós candidatos e nós cortados pelo *bound* ao longo da busca.

A Figura 4.5 mostra a convergência dos valores dos limites no decorrer da elaboração do plano. Note que o Limite Superior é o que sofre a menor variação, já que a função permite um cálculo mais preciso, considerando os VPLs máximos dos módulos não escalonados.

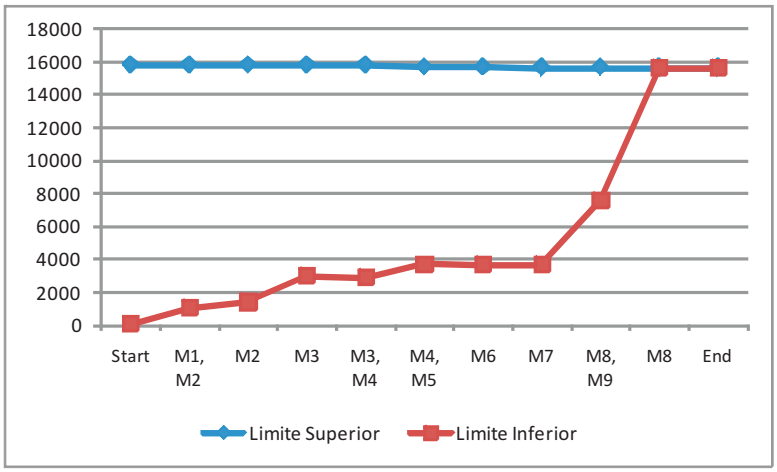

Figura 4.5: Convergência dos limites superior e inferior ao longo da construção do plano.

4.6 Desempenho da Busca

No decorrer da busca foram realizadas 15 ramificações e a árvore utilizou apenas 30 nós. A árvore da busca exaustiva, sem o corte do *bound*, teria utilizado 22.212 nós para encontrar a mesma solução. Isso significa que, graças ao mecanismo de *bound*, foi necessário gerar apenas 0,14% dos nós para enconrar o plano ótimo.

A função de Limite Superior é uma função decrescente, já que o valor estimado de forma otimista converge para um valor real à medida em que os módulos são escalonados, mas a função de Limite Inferior pode não ser crescente, pois o desenvolvimento de módulos com duração maior que 1 período pode ocasionar custos periódicos que só serão compensados após sua conclusão. No entanto, isso não prejudica a busca, pois o critério de corte é dado por *MaiorLI*, que contém o maior valor de Limite Inferior encontrado até o momento e, portanto, é uma função crescente.

4.7 Criando Cenários Alternativos

Os consultores estratégicos da empresa solicitaram que se tentasse elaborar um plano de apenas 7 meses para o projeto, de forma que a *LRU* possa ser a primeira a oferecer esse tipo de serviço e ainda tenha tempo suficiente para conquistar uma boa posição no mercado.

A árvore obtida ao final da busca, para um prazo de 7 meses, é apresentada na Seção A.2 do Apêndice. Ela indica que o melhor plano para esse cenário é:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4)$$

Esse plano fornece o VPL máximo de $3.831 mil, que é apenas 24,5% do valor anterior de $15.642 mil e corresponde ao desenvolvimento de apenas 4 dos 9 módulos do projeto ao longo de 5 meses.

O líder do projeto, preocupado com o fato de não ser possível concluir todos os módulos no prazo estabelecido, decidiu realizar outra simulação para saber qual seria o VPL do projeto caso houvesse mais tempo disponível. Com isso, ele descobriu que se o prazo fosse ampliado em 1 mês, ou seja, se houvessem 8 meses para o desenvolvimento do projeto, seu VPL máximo seria de $7.596 mil. A árvore de busca que apresenta esse resultado é apresentada na Seção A.3 do Apêndice. O plano alternativo que conduz a esse VPL é:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M9)$$

O significativo aumento do VPL de $3.831 para $7.596 pôde ser obtido graças ao desenvolvimento de M9 e dos elementos arquiteturais que o precedem (M5, M6 e M7).

Após apresentar as propostas de desenvolvimento com prazos de 7, 8 e 9 meses aos patrocinadores do projeto foi levantada a hipótese de se realizar o desenvolvimento em 9 meses, mas com um capital inicial de apenas $80 mil reais, ao invés dos $100 mil oferecidos inicialmente.

Com base nas novas condições propostas, construiu-se a árvore de busca, apresentada na Seção A.4 do Apêndice, para encontrar o melhor plano que atenda a tais restrições. O plano obtido ao final da busca foi:

$$(M1) \rightarrow (M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M9)$$

Esse plano possui um VPL de $6.575 mil. O valor é maior que os $3.831 mil do plano de 7 meses, porém menor que os valores estimados para 8 ou 9 meses de prazo com capital inicial de $100 mil. O motivo de o valor ser menor é que, devido ao baixo orçamento, não é possível iniciar o desenvolvimento dos módulos M1 e M2 paralelamente. Isso acarreta um adiamento em todas as atividades sucessoras e, conseqüentemente, uma redução nos seus respectivos VPLs, além de não deixar tempo suficiente para a conclusão de M8.

4.8 Comparando os Cenários Propostos

A Tabela 4.5 apresenta um quadro comparativo dos diferentes cenários analisados e das soluções encontradas.

Cenário	Prazo Máximo	Capital Inicial	VPL Máximo	Escopo do Plano Encontrado	Duração do Desenvolvimento
1	15 meses	$100	$15.642	9 módulos	9 meses
2	7 meses	$100	$3.831	4 módulos	5 meses
3	8 meses	$100	$7.596	8 módulos	8 meses
4	9 meses	$80	$6.575	8 módulos	9 meses

Tabela 4.5: Comparação entre os cenários propostos.

Note que o cenário 1, além de ser o que apresenta o maior VPL, é o único em que o plano contempla todo o escopo do projeto. Nesse cenário, apesar do prazo máximo ser de 15 meses, o tempo necessário para o desenvolvimento do projeto e obtenção do VPL máximo é de apenas 9 meses. Nas outras alternativas, as restrições de prazo e de capital inviabilizam o desenvolvimento de todo o projeto.

5 *Conclusões*

A seguir, são apresentadas algumas respostas aos principais questionamentos acerca do método proposto e as conclusões deste trabalho.

5.1 Discussão

No decorrer do texto, foi apresentada a proposta de um método capaz de fornecer resultados melhores que o IFM para o problema de maximização do VPL de um projeto. A seguir, respondemos algumas questões relevantes sobre esse problema, a solução proposta e as implicações do seu uso.

5.1.1 Qual é a importância das questões apontadas pelo IFM?

Desde a sua publicação em 2004, as idéias de Denne e Cleland-Huang sobre *Minimum Marketable Features* e o *Incremental Funding Method* têm influenciado o trabalho não somente de outros pesquisadores mas também dos desenvolvedores de software em todo o mundo.

Entre os trabalhos de pesquisa, podemos citar as teses de mestrado de Carlos Eduardo Mendes de Azevedo (AZEVEDO, 2007) e Breno Peixoto Barbosa (BARBOSA, 2008), que tratam o problema de encontrar a melhor ordem de desenvolvimento considerando outros aspectos da realidade e com o uso de outras abordagens.

Uma busca no Google Acadêmico (scholar.google.com.br) revela que mais de duas dezenas de artigos já foram publicados sobre o tema, propondo novos usos e aperfeiçoamentos para as idéias de Denne e Cleland-Huang.

A busca no Google comum (www.google.com) revela a existência de mais de 1.500 sites sobre o assunto, contendo comentários, críticas, ferramentas para facilitar o uso de MMFs e casos de sucesso do mundo real.

Entre as ferramentas de apoio, podemos citar o ReleasePlanner (www.releaseplanner.com),

que implementa a abordagem apresentada por Ruhe e Saliu, capaz de considerar os interesses de diferentes partes na elaboração do plano de desenvolvimento (RUHE; SALIU, 2005).

É possível que, no decorrer do tempo, as idéias de Denne e Cleland-Huang e suas inúmeras variantes venham a se revelar como um verdadeiro marco no desenvolvimento de software, colocando essa atividade no centro da criação de valor e de vantagens competitivas em organizações de todos os tipos e tamanhos.

5.1.2 Qual é a complexidade do problema de encontrar o plano de desenvolvimento que fornece o maior VPL?

O problema de seqüenciamento abordado pelo IFM pertence a uma categoria de problemas computacionais para a qual não existem soluções algorítmicas polinomiais conhecidas. É necessário encontrar e comparar todas as combinações possíveis. Esse tipo de varredura exaustiva geralmente requer uma enorme quantidade de tempo para ser executada em um computador (WANG, 1997; WEISS, 1999).

Observando o caso em que não existem precedências entre os módulos, podemos calcular o número de seqüências possíveis como sendo da ordem do fatorial de *n*, ou *O(n!)*. A Tabela 5.1 mostra a progressão do número de seqüências completas que se pode obter em um projeto dividido em módulos. Note que o número de sequências cresce rapidamente, já que

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1.$$

Nº Módulos	Nº de Seqüencias
1	1
2	2
3	6
4	24
5	120
6	720
⋮	⋮
20	2.432.902.008.176.640.000

Tabela 5.1: Quantidade de seqüências completas que se pode obter em um projeto dividido em módulos.

No entanto, devido à possibilidade de adiamento do desenvolvimento dos módulos e ao fato de que planos de desenvolvimento parciais também podem ser considerados soluções válidas, o número de soluções consideradas pelo método proposto tende a ser ainda maior.

5.1.3 Por que utilizar um método Branch & Bound para solucionar o problema de maximização do VPL?

As principais entradas desse problema são o conjunto de módulos que constituem o projeto e as relações de dependência que existem entre esses módulos, e a quantidade de seqüências válidas de desenvolvimento para os módulos de um projeto de *software* tende a crescer exponencialmente com o número de módulos em que o projeto está dividido. Conseqüentemente, só é viável enumerar em tempo hábil todas estas seqüências quando o projeto está dividido em poucos módulos. Isto faz com que o problema de encontrar a seqüência de módulos que maximiza o VPL de um projeto seja uma boa oportunidade para o emprego de métodos heurísticos como o *Branch & Bound*, que, na grande maioria dos casos, não precisa enumerar todas as seqüências para encontrar a solução ótima (LIBERTI, 2003).

5.1.4 Como o Branch & Bound garante a solução ótima?

Segundo (DEMEULEMEESTER; HERROELEN, 2002), são dois os requisitos para que a solução encontrada por um método *Branch & Bound* similar ao apresentado seja ótima. O primeiro é que o procedimento de *branch* deve cobrir todo o espaço de soluções, garantindo que a solução ótima seja gerada. O atendimento a esse requisito ocorre quando consideramos, a cada período, todas as combinações de atividades que podem estar em execução, conforme descrito na função de *branch* apresentada na Seção 3.5.

O segundo requisito para que a solução encontrada seja ótima é que as heurísticas devem ser admissíveis, ou seja, o valor da melhor solução encontrada a partir de um determinado nó da árvore de busca não pode ser menor do que o valor indicado pelo seu limite inferior, nem maior do que o valor de seu limite superior. Com isso, evita-se que a solução ótima seja descartada indevidamente. Uma vez que a função de limite superior apresentada, na Seção 3.3, é uma heurística otimista e a função de limite inferior, apresentada na Seção 3.4, é uma heurística pessimista, esse requisito também é atendido.

5.1.5 O que este método tem de melhor se comparado ao IFM?

Existem duas grandes vantagens na utilização do método *Branch & Bound* em substituição à Heurística IFM. A primeira vantagem é que este método garante uma solução ótima para o problema, enquanto que a heurística do IFM se contenta com resultados inferiores. Em projetos que custem milhões de reais, uma diferença de 10% significa algumas centenas de milhares de reais. Desperdícios desta magnitude representam, em termos absolutos, um desempenho con-

sideravelmente longe do ótimo, tendo como conseqüência a perda potencial de competitividade, dando margem ao crescimento da concorrência.

A segunda vantagem é que o método pode ser aplicado a projetos que apresentem relações mais complexas entre as unidades de software, aceitando os casos que elas dependam de mais de uma outra unidade e que tenham restrições de recursos (financeiros e não financeiros). Embora estas sejam situações comuns no universo dos projetos, tal construção não é suportada pela heurística do IFM em tempo polinomial.

5.1.6 Qual o impacto esperado nas negociações das propostas de desenvolvimento?

Diante de um processo de apresentação de propostas para o desenvolvimento de sistemas, os fornecedores de serviços de TI que fizerem uso do método proposto nesta dissertação terão uma nítida vantagem competitiva sobre os demais, já que suas propostas serão aquelas que, com certeza, farão com que o projeto tenha o maior valor VPL.

Por outro lado, os fornecedores de serviço que tiverem optado por utilizar o IFM estarão se contentando, no caso geral, apenas com uma solução aproximada. Não podendo, portanto, declarar que estão seguramente oferecendo, do ponto de vista financeiro, a melhor solução para seus clientes.

5.1.7 Futuramente, que melhorias poderiam ser realizadas no método?

Os problemas do mundo real apresentam um grande número de variáveis e detalhes, o que dificulta a construção de um único método capaz de considerar todas as características e situações que se pode encontrar em um projeto real. A seguir estão listadas algumas características do mundo real que ainda não foram incorporadas ao método proposto:

- O tratamento de incertezas quanto aos custos, benefícios e durações das atividades;
- Considerar que um mesmo módulo possa ser executada de duas formas diferentes, por exemplo a execução em dois períodos alocando uma única pessoa para o trabalho ou a execução em um único período alocando duas pessoas;
- Considerar a existência de módulos que não sejam essenciais para o projeto e módulos mutuamente exclusivos;

- Permitir que o desenvolvimento de um módulo comece antes que todos os seus antecessores terminem;
- Considerar a possibilidade de um mesmo recurso assumir diferentes papéis na execução das atividades. Por exemplo: um programador senior pode exercer o papel de um programador junior.

5.2 Conclusão

Este trabalho apresentou um método *Branch & Bound* para encontrar um plano de desenvolvimento que maximize o valor de um projeto de *software* para as organizações.

O uso do método pode causar um impacto bastante positivo nas negociações entre fornecedores de serviços de TI e seus respectivos clientes, aumentando as chances do projeto ser contratado.

Este método também pode ser utilizado como ferramenta de apoio ao gerente, auxiliando na decisão de quando cada módulo deve ser implementado mesmo após mudanças no cenário do projeto.

Após comparar o método apresentado com um outro método, conhecido como IFM, o uso do *Branch & Bound* demonstrou ser mais vantajoso por fornecer uma solução ótima e por permitir restrições mais flexíveis às relações de precedência que possam existir entre as unidades do *software*, o que o torna mais adequado às diversas estruturas de interdependência que as unidades dos projetos podem apresentar no mundo real.

Referências Bibliográficas

ABACUS, A.; BARKER, M.; FREEDMAN, P. Using test-driven software development tools. *Software, IEEE*, v. 22, n. 2, p. 88–91, March-April 2005.

ABES. *Mercado Brasileiro de Software: panorama e tendências - 2009*. 1ª. São Paulo, SP, Brazil, 2009.

ALENCAR, A. J. et al. Maximizing the business value of software projects: A branch & bound approach. In: CORDEIRO, J.; FILIPE, J. (Ed.). *10th International Conference on Enterprise Information Systems (ICEIS)*. Barcelona, Espanha: Institute for Systems and Technologies of Information, Control and Communication, 2008. ISAS-2, p. 162–169.

ALENCAR, A. J. et al. Um método branch & bound para maximizar o valor de projetos de software. In: ARAúJO, R. M. de; CIDRAL, A. (Ed.). *Anais SBSI 2008 - IV Simpósio Brasileiro de Sistemas de Informação*. Porto Alegre, RS, Brazil: Sociedade Brasileira de Computação (SBC), 2008. p. 128–139.

AMIT, R.; ZOTT, C. Value creation in e-business. *Strategic Management Journal*, v. 22, n. 6/7, p. 493 – 520, June-July 2001.

AZEVEDO, C. E. M. de. *Um Método para a Determinação do Máximo Retorno Financeiro em Projetos de Software usando Modelos Estocásticos*. Dissertação (Mestrado) — PPGI/IM/NCE-Universidade Federal do Rio de Janeiro, February 26^{th} 2007.

BARBOSA, B. P. *Gerando Políticas de Investimentos para Projetos de Software em Ambientes Incertos*. Dissertação (Mestrado) — PPGI/IM/NCE-Universidade Federal do Rio de Janeiro, February 28^{th} 2008.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Information availiable in the Internet at www.agilemanifesto.org. Site last visited on March 21^{st}, 2007.

DATTA, S. *Six Sigma software development*. Boca Raton, FL, USA: Auertbch, 2007.

DEMEULEMEESTER, E. L.; HERROELEN, W. S. *Project Scheduling: A Research Handbook*. 1^{st} edition. New York, NY, USA: Springer, 2002. (International Series in Operations Research & Management Science).

DENNE, M.; CLELAND-HUANG, J. The incremental funding method: data-driven software development. *IEEE Software*, v. 21, n. 3, p. 39–47, May-June 2004.

DENNE, M.; CLELAND-HUANG, J. *Software by Numbers - Low-Risk, High-Return Development*. Upper Saddle River, NJ, USA: Prentice Hall, 2004.

DENNE, M.; CLELAND-HUANG, J. Financially informed requirements prioritization. In: ROMAN, G.-C.; GRISWOLD, W.; NUSEIBEH, B. (Ed.). 27^{th} *international conference on Software Engineering*. St. Louis, MO, USA: ACM, 2005. p. 710–711.

FABOZZI, F. J.; DAVIS, H. A.; CHOUDHRY, M. *Introduction to Structured Finance*. San Francisco, USA: John Wiley, 2006.

GROSS, J. L.; YELLEN, J. *Graph Theory and Its Applications*. 2^{nd}. Boca Raton, FL, USA: Chapman & Hall and CRC, 2005.

HELO, P.; HILMOLA, O.-P.; MAUNUKSELA, A. Managing the productivity of product development: a system dynamics analysis. *International Journal of Management and Enterprise Development*, v. 1, n. 4, p. 333–344, 2004.

HIBBS, C.; JEWETT, S.; SULLIVAN, M. *The Art of Lean Software Development: A Practical and Incremental Approach*. Sebastopol, CA, USA: O'Reilly Media, 2009.

HIGHSMITH, J. *Agile Software Development Ecosystems*. Upper Saddle River, NJ, USA: Addison Wesley, 2002.

HILLIER, F. S.; LIEBERMAN, G. J. *Introduction to operations research*. 7^{th}. New York, NY: McGraw-Hill, 2001.

HUBBARD, D. W. *How to Measure Anything: Finding the Value of "Intangibles"in Business*. Hoboken, NJ, USA: John Wiley, 2007.

HUI, A. K. T.; LIU, D. B. A bayesian belief network model and tool to evaluate risk and impact in software development projects. In: ELECTRICAL, I. of; (IEEE), E. E. (Ed.). *Reliability and Maintainability, 2004 Annual Symposium - RAMS*. Los Angeles, CA, USA: Collegis/Loyola Marymount Univ., 2004. p. 297– 301.

JORGENSON, D. W.; HO, M. S.; STIROH, K. J. Growth of us industries and investments in information technology and higher education. *Economic Systems Research*, v. 15, n. 3, p. 279–325, September 2003.

LAM, H. New design-to-test software strategies accelerate time-to-market. In: 29^{th} *International Electronics Manufacturing Technology Symposium*. San Jose, CA, USA: IEEE, 2004. p. 140–143.

LAPLANTE, P. A. *What every engineer should know about software engineering*. Boca Raton, FL, USA: CRC Press, 2007.

LARMAN, C. *Agile and iterative development: a manager's guide*. Boston, MA, USA: Pearson Education, 2004.

LIBERTI, L. Optimization and optimal control. In: ____. Hackensack, NJ, USA: World Scientific, 2003. (Computers and Operations Research, v. 1), cap. Comparison of Convex Relaxations for Monomials of Odd Degree, p. 165–174.

MCMANUS, J. C. *Risk Management in Software Development Projects*. Maryland Heights, MO, USA: Elsevier, 2003.

NORD, R.; TOMAYKO, J. Software architecture-centric methods and agile development. *Software, IEEE*, v. 23, n. 2, p. 47–53, March-April 2006.

RASHID, A.; MOREIRA, A.; ARAúJO, J. Modularisation and composition of aspectual requirements. In: NORTHEASTERN UNIVERSITY. *Proceedings of the 2nd International Conference on Aspect-oriented Software Development*. Boston, Massachusetts, USA: ACM, 2003. p. 11 – 20.

RUHE, G.; SALIU, M. O. The art and science of software release planning. *Software, IEEE*, v. 22, n. 6, p. 47–53, November-December 2005.

STEINDL, C. From agile software development to agile businesses. In: MATOS, J. S.; CRNKOVIC, I. (Ed.). 31st *EUROMICRO Conference on Software Engineering and Advanced Applications*. Porto, Portugal, 2005. p. 258– 265.

STINSON EDWARD W. DAVIS, B. M. K. J. P. Multiple resource-constrained scheduling using branch and bounds. *AIIE Transactions*, v. 10, n. 3, p. 252–259, September 1978.

TAYNTOR, C. B. *Matrics-driven Enterprise software development: effectively meeting evolving business needs*. Fort Lauderdale, FL, USA: J. Ross, 2007.

WANG, J. *Average-Case Intractable NP Problems - Advances in Languages, Algorithms, and Complexity*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

WEISS, M. A. *Data Structures and Algorithm Analysis in C++ (2nd ed.)*. Upper Saddle River, NJ, USA: Addison Wesley, 1999.

WHITTLE, R.; MYRICK, C. B. *Enterprise Business Architecture*. Boca Raton, FL, USA: Auerbach, 2005.

APÊNDICE A – Árvores Geradas

A seguir, apresentamos a evolução da árvore de busca ao longo das ramificações que ocorrem durante a execução do método para o primeiro cenário e as árvores finais, obtidas ao término da execução do algorítimo, para cada um dos cenários alternativos.

A marcação *branched*, indica que os nós que foram expandidos, enquanto a marcação *bounded* indica os que foram descartados no decorrer da busca.

A.1 Cenário 1: Prazo de 15 meses e R$ 100.000,00 de capital

1º *Branch* - Expansão do nó #0

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched> *
.       1:  {1}[100, 12.984] {P:3|$:100|T:11}
.       2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11}
.       3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11}
.       4:  {1-M2}[51, 15.730] {P:2|$:3|T:11}
MaiorLI = 1.124
Candidatos = {1, 2, 3, 4}
```

2º *Branch* - Expansão do nó #3

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched>
.       1:  {1}[100, 12.984] {P:3|$:100|T:11}
.       2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11}
.       3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11} <Branched> *
.       .       5:  {2-M2}[1.419, 15.786] {P:3|$:87|T:10}
.       4:  {1-M2}[51, 15.730] {P:2|$:3|T:11}
MaiorLI = 1.419
Candidatos = {1, 2, 4, 5}
```

3º *Branch* - Expansão do nó #5

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched>

.       1:  {1}[100, 12.984] {P:3|$:100|T:11}

.       2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11}

.       3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11} <Branched>

.       .       5:  {2-M2}[1.419, 15.786] {P:3|$:87|T:10} <Branched> *

.       .       .       6:  {3}[1.419, 13.061] {P:3|$:205|T:9}

.       .       .       7:  {3-M3}[3.000, 15.786] {P:3|$:213|T:9}

.       4:  {1-M2}[51, 15.730] {P:2|$:3|T:11}

MaiorLI = 3.000

Candidatos = {1, 2, 4, 6, 7}
```

4º *Branch* - Expansão do nó #7

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched>

.       1:  {1}[100, 12.984] {P:3|$:100|T:11}

.       2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11}

.       3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11} <Branched>

.       .       5:  {2-M2}[1.419, 15.786] {P:3|$:87|T:10} <Branched>

.       .       .       6:  {3}[1.419, 13.061] {P:3|$:205|T:9}

.       .       .       7:  {3-M3}[3.000, 15.786] {P:3|$:213|T:9} <Branched> *

.       .       .       .       8:  {4}[3.000, 13.223] {P:3|$:448|T:8}

.       .       .       .       9:  {4-M4}[2.917, 15.786] {P:2|$:284|T:8}

.       .       .       .       10:  {4-M4,M5}[2.787, 15.657] {P:2|$:154|T:8}

.       .       .       .       11:  {4-M5}[2.870, 13.094] {P:3|$:319|T:8}

.       4:  {1-M2}[51, 15.730] {P:2|$:3|T:11}

MaiorLI = 3.000

Candidatos = {1, 2, 4, 6, 8, 9, 10, 11}
```

5º *Branch* - Expansão do nó #9

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched>

.       1:  {1}[100, 12.984] {P:3|$:100|T:11}

.       2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11}

.       3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11} <Branched>

.       .       5:  {2-M2}[1.419, 15.786] {P:3|$:87|T:10} <Branched>

.       .       .       6:  {3}[1.419, 13.061] {P:3|$:205|T:9}

.       .       .       7:  {3-M3}[3.000, 15.786] {P:3|$:213|T:9} <Branched>

.       .       .       .       8:  {4}[3.000, 13.223] {P:3|$:448|T:8}

.       .       .       .       9:  {4-M4}[2.917, 15.786] {P:2|$:284|T:8} <Branched> *

.       .       .       .       .       12:  {5-M4}[3.831, 13.334] {P:3|$:648|T:7}

.       .       .       .       .       13:  {5-M4,M5}[3.704, 15.660] {P:3|$:521|T:7}

.       .       .       .       10:  {4-M4,M5}[2.787, 15.657] {P:2|$:154|T:8}

.       .       .       .       11:  {4-M5}[2.870, 13.094] {P:3|$:319|T:8}

.       4:  {1-M2}[51, 15.730] {P:2|$:3|T:11}

MaiorLI = 3.831

Candidatos = {1, 2, 4, 6, 8, 10, 11, 12, 13}
```

15º *Branch* - Expansão do nó #29

```
0:  {1-Start}[100, 15.786] {P:3|$:100|T:12} <Branched>
.  1:  {1}[100, 12.984] {P:3|$:100|T:11} <BOUNDED>
.  2:  {1-M1}[1.124, 13.040] {P:3|$:99|T:11} <BOUNDED>
.  3:  {1-M1,M2}[1.075, 15.786] {P:2|$:2|T:11} <Branched>
.  .  5:  {2-M2}[1.419, 15.786] {P:3|$:87|T:10} <Branched>
.  .  .  6:  {3}[1.419, 13.061] {P:3|$:205|T:9} <BOUNDED>
.  .  .  7:  {3-M3}[3.000, 15.786] {P:3|$:213|T:9} <Branched>
.  .  .  .  8:  {4}[3.000, 13.223] {P:3|$:448|T:8} <BOUNDED>
.  .  .  .  9:  {4-M4}[2.917, 15.786] {P:2|$:284|T:8} <Branched>
.  .  .  .  .  12:  {5-M4}[3.831, 13.334] {P:3|$:648|T:7} <BOUNDED>
.  .  .  .  .  13:  {5-M4,M5}[3.704, 15.660] {P:3|$:521|T:7} <Branched>
.  .  .  .  .  .  15:  {6}[3.704, 13.207] {P:3|$:887|T:6} <BOUNDED>
.  .  .  .  .  .  16:  {6-M6}[3.695, 15.651] {P:3|$:878|T:6} <Branched>
.  .  .  .  .  .  .  20:  {7}[3.695, 13.198] {P:3|$:1.262|T:5} <BOUNDED>
.  .  .  .  .  .  .  21:  {7-M7}[3.687, 15.642] {P:3|$:1.253|T:5} <Branched>
.  .  .  .  .  .  .  .  24:  {8}[3.687, 13.190] {P:3|$:1.630|T:4} <BOUNDED>
.  .  .  .  .  .  .  .  25:  {8-M8}[3.674, 14.932] {P:2|$:1.604|T:4} <BOUNDED>
.  .  .  .  .  .  .  .  26:  {8-M8,M9}[7.583, 15.642] {P:2|$:1.813|T:4} <Branched>
.  .  .  .  .  .  .  .  .  28:  {9-M8}[15.642, 15.642] {P:3|$:3.113|T:3} <Branched>
.  .  .  .  .  .  .  .  .  .  29:  {10-End}[15.642, 15.642] {P:3|$:3.113|T:3} <Branched> *
.  .  .  .  .  .  .  .  27:  {8-M9}[7.596, 13.900] {P:3|$:1.838|T:4} <BOUNDED>
.  .  .  .  10:  {4-M4,M5}[2.787, 15.657] {P:2|$:154|T:8} <Branched>
.  .  .  .  .  17:  {5-M4}[3.702, 15.657] {P:3|$:518|T:7} <Branched>
.  .  .  .  .  .  18:  {6}[3.702, 13.205] {P:3|$:884|T:6} <BOUNDED>
.  .  .  .  .  .  19:  {6-M6}[3.693, 15.648] {P:3|$:875|T:6} <Branched>
.  .  .  .  .  .  .  22:  {7}[3.693, 13.196] {P:3|$:1.259|T:5} <BOUNDED>
.  .  .  .  .  .  .  23:  {7-M7}[3.684, 15.640] {P:3|$:1.250|T:5} <BOUNDED>
.  .  .  .  11:  {4-M5}[2.870, 13.094] {P:3|$:319|T:8} <BOUNDED>
.  4:  {1-M2}[51, 15.730] {P:2|$:3|T:11} <Branched>
.  .  14:  {2-M2}[394, 12.912] {P:3|$:22|T:10} <BOUNDED>
MaiorLI = 15.642
Candidatos = {}
```

O melhor plano, que apresenta um VPL de $15.642 mil mil, é o indicado pelo nó 29:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M8, M9) \rightarrow (M8)$$

A.2 Cenário 2: Prazo de 7 meses e R$ 100.000,00 de capital

```
0:  {1-Start}[100, 3.831] {P:3|$:100|T:7} <Branched>
.  1:  {1}[100, 3.481] {P:3|$:100|T:6} <BOUNDED>
.  2:  {1-M1}[1.124, 3.538] {P:3|$:99|T:6} <BOUNDED>
.  3:  {1-M1,M2}[1.075, 3.831] {P:2|$:2|T:6} <Branched>
.  .  5:  {2-M2}[1.419, 3.831] {P:3|$:87|T:5} <Branched>
.  .  .  6:  {3}[1.419, 3.558] {P:3|$:205|T:4} <BOUNDED>
.  .  .  7:  {3-M3}[3.000, 3.831] {P:3|$:213|T:4} <Branched>
.  .  .  .  8:  {4}[3.000, 3.720] {P:3|$:448|T:3} <BOUNDED>
.  .  .  .  9:  {4-M4}[2.917, 3.831] {P:2|$:284|T:3} <Branched>
.  .  .  .  .  12:  {5-M4}[3.831, 3.831] {P:3|$:648|T:2} <Branched>
.  .  .  .  .  .  14:  {6}[3.831, 3.831] {P:3|$:1.013|T:1} <Branched>
.  .  .  .  .  .  .  16:  {7}[3.831, 3.831] {P:3|$:1.397|T:0} <Branched> *
.  .  .  .  .  .  .  17:  {7-M5}[3.709, 3.709] {P:3|$:1.276|T:0} <BOUNDED>
.  .  .  .  .  .  15:  {6-M5}[3.707, 3.707] {P:3|$:889|T:1} <BOUNDED>
.  .  .  .  .  13:  {5-M4,M5}[3.704, 3.704] {P:3|$:521|T:2} <BOUNDED>
.  .  .  .  10:  {4-M4,M5}[2.787, 3.702] {P:2|$:154|T:3} <BOUNDED>
.  .  .  .  11:  {4-M5}[2.870, 3.591] {P:3|$:319|T:3} <BOUNDED>
.  4:  {1-M2}[51, 3.775] {P:2|$:3|T:6} <BOUNDED>
MaiorLI = 3.831
Candidatos = {}
```

O melhor plano, que apresenta um VPL de $3.831 mil, é o indicado pelo nó 16:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4)$$

A.3 Cenário 3: Prazo de 8 meses e R$ 100.000,00 de capital

```
0:  {1-Start}[100, 7.740] {P:3|$:100|T:8} <Branched>
.  1:  {1}[100, 3.481] {P:3|$:100|T:7} <BOUNDED>
.  2:  {1-M1}[1.124, 3.538] {P:3|$:99|T:7} <BOUNDED>
.  3:  {1-M1,M2}[1.075, 7.740] {P:2|$:2|T:7} <Branched>
.  .  5:  {2-M2}[1.419, 7.740] {P:3|$:87|T:6} <Branched>
.  .  .  6:  {3}[1.419, 3.558] {P:3|$:205|T:5} <BOUNDED>
.  .  .  7:  {3-M3}[3.000, 7.740] {P:3|$:213|T:5} <Branched>
.  .  .  .  8:  {4}[3.000, 3.720] {P:3|$:448|T:4} <BOUNDED>
.  .  .  .  9:  {4-M4}[2.917, 7.740] {P:2|$:284|T:4} <Branched>
.  .  .  .  .  12:  {5-M4}[3.831, 3.831] {P:3|$:648|T:3} <BOUNDED>
.  .  .  .  .  13:  {5-M4,M5}[3.704, 7.613] {P:3|$:521|T:3} <Branched>
.  .  .  .  .  .  15:  {6}[3.704, 3.704] {P:3|$:887|T:2} <BOUNDED>
.  .  .  .  .  .  16:  {6-M6}[3.695, 7.604] {P:3|$:878|T:2} <Branched>
.  .  .  .  .  .  .  20:  {7}[3.695, 3.695] {P:3|$:1.262|T:1} <BOUNDED>
.  .  .  .  .  .  .  21:  {7-M7}[3.687, 7.596] {P:3|$:1.253|T:1} <Branched>
.  .  .  .  .  .  .  .  24:  {8}[3.687, 3.687] {P:3|$:1.630|T:0} <BOUNDED>
.  .  .  .  .  .  .  .  25:  {8-M9}[7.596, 7.596] {P:3|$:1.838|T:0} <Branched> *
.  .  .  .  10:  {4-M4,M5}[2.787, 7.611] {P:2|$:154|T:4} <Branched>
.  .  .  .  .  17:  {5-M4}[3.702, 7.611] {P:3|$:518|T:3} <Branched>
.  .  .  .  .  .  18:  {6}[3.702, 3.702] {P:3|$:884|T:2} <BOUNDED>
.  .  .  .  .  .  19:  {6-M6}[3.693, 7.602] {P:3|$:875|T:2} <Branched>
.  .  .  .  .  .  .  22:  {7}[3.693, 3.693] {P:3|$:1.259|T:1} <BOUNDED>
.  .  .  .  .  .  .  23:  {7-M7}[3.684, 7.593] {P:3|$:1.250|T:1} <BOUNDED>
.  .  .  .  11:  {4-M5}[2.870, 3.591] {P:3|$:319|T:4} <BOUNDED>
.  4:  {1-M2}[51, 7.683] {P:2|$:3|T:7} <Branched>
.  .  14:  {2-M2}[394, 3.410] {P:3|$:22|T:6} <BOUNDED>
MaiorLI = 7.596
Candidatos = {}
```

O melhor plano, que apresenta um VPL de $7.596 mil, é o indicado pelo nó 25:

$$(M1, M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M9)$$

A.4 Cenário 4: Prazo de 9 meses e R$ 80.000,00 de capital

```
0:  {1-Start}[80, 15.766] {P:3|$:80|T:9} <Branched>
.  1:  {1}[80, 6.660] {P:3|$:80|T:8} <Branched>
.  .  15:  {2}[80, 3.067] {P:3|$:80|T:7} <BOUNDED>
.  .  16:  {2-M1}[1.048, 3.159] {P:3|$:79|T:7} <BOUNDED>
.  .  17:  {2-M2}[32, 6.568] {P:2|$:-15|T:7} <BOUNDED>
.  2:  {1-M1}[1.104, 6.716] {P:3|$:79|T:8} <Branched>
.  .  4:  {2}[1.104, 3.215] {P:3|$:145|T:7} <BOUNDED>
.  .  5:  {2-M2}[1.056, 6.716] {P:2|$:50|T:7} <Branched>
.  .  .  6:  {3-M2}[1.378, 6.716] {P:3|$:161|T:6} <Branched>
.  .  .  .  7:  {4}[1.378, 3.250] {P:3|$:277|T:5} <BOUNDED>
.  .  .  .  8:  {4-M3}[2.797, 6.716] {P:3|$:284|T:5} <Branched>
.  .  .  .  .  9:  {5}[2.797, 3.409] {P:3|$:515|T:4} <BOUNDED>
.  .  .  .  .  10:  {5-M4}[2.716, 6.716] {P:2|$:354|T:4} <Branched>
.  .  .  .  .  .  13:  {6-M4}[3.518, 3.518] {P:3|$:711|T:3} <BOUNDED>
.  .  .  .  .  .  14:  {6-M4,M5}[3.393, 6.592] {P:3|$:586|T:3} <Branched>
.  .  .  .  .  .  .  18:  {7}[3.393, 3.393] {P:3|$:945|T:2} <BOUNDED>
.  .  .  .  .  .  .  19:  {7-M6}[3.385, 6.583] {P:3|$:936|T:2} <Branched>
.  .  .  .  .  .  .  .  23:  {8}[3.385, 3.385] {P:3|$:1.313|T:1} <BOUNDED>
.  .  .  .  .  .  .  .  24:  {8-M7}[3.376, 6.575] {P:3|$:1.304|T:1} <Branched>
.  .  .  .  .  .  .  .  .  27:  {9}[3.376, 3.376] {P:3|$:1.673|T:0} <BOUNDED>
.  .  .  .  .  .  .  .  .  28:  {9-M9}[6.575, 6.575] {P:3|$:1.878|T:0} <Branched> *
.  .  .  .  .  11:  {5-M4,M5}[2.589, 6.589] {P:2|$:227|T:4} <Branched>
.  .  .  .  .  .  20:  {6-M4}[3.391, 6.589] {P:3|$:584|T:3} <Branched>
.  .  .  .  .  .  .  21:  {7}[3.391, 3.391] {P:3|$:942|T:2} <BOUNDED>
.  .  .  .  .  .  .  22:  {7-M6}[3.382, 6.581] {P:3|$:934|T:2} <Branched>
.  .  .  .  .  .  .  .  25:  {8}[3.382, 3.382] {P:3|$:1.310|T:1} <BOUNDED>
.  .  .  .  .  .  .  .  26:  {8-M7}[3.374, 6.572] {P:3|$:1.302|T:1} <BOUNDED>
.  .  .  .  .  12:  {5-M5}[2.670, 3.282] {P:3|$:388|T:4} <BOUNDED>
.  3:  {1-M2}[31, 15.710] {P:2|$:-17|T:8} <Branched>
MaiorLI = 6.575
Candidatos = {}
```

O melhor plano, que apresenta um VPL de $6.575 mil, é o indicado pelo nó 28:

$$(M1) \rightarrow (M2) \rightarrow (M2) \rightarrow (M3) \rightarrow (M4) \rightarrow (M4, M5) \rightarrow (M6) \rightarrow (M7) \rightarrow (M9)$$

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CCMN - Bloco C - Cidade Universitária - Ilha do Fundão
Rio de Janeiro - RJ CEP: 21941-916
www.ppgi.ufrj.br