PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

Isac Mendes Lacerda

Análise Empírica de Algoritmos para Escalonamento de Projetos com Maximização de Valor Presente Líquido

TESE DE DOUTORADO

Instituto de Matemática

Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais

Isac Mendes Lacerda

Análise Empírica de Algoritmos para Escalonamento de Projetos com Maximização de Valor Presente Líquido

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais da Universidade Federal do Rio de Janeiro, como parte dos requisitos à obtenção do título de Doutor em Informática.

Orientadores:
Jayme Luiz Szwarcfiter, Ph.D.
Eber Assis Schmitz, Ph.D.
Rosiane de Freitas Rodrigues, Ph.D.

Rio de Janeiro
2022

CIP - Catalogação na Publicação

Lacerda, Isac Mendes
L131a Análise empírica de algoritmos para escalonamento de projetos com maximização de valor presente líquido / Isac Mendes Lacerda. -- Rio de Janeiro, 2022.
109 f.

Orientador: Jayme Luiz Szwarcfiter.
Tese (doutorado) - Universidade Federal do Rio de Janeiro, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Programa de Pós-Graduação em informática, 2022.

1. Algoritmos de escalonamento de projetos. 2. Maximização de valor presente líquido. 3. Análise empírica. I. Szwarcfiter, Jayme Luiz, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

Análise Empírica de Algoritmos para Escalonamento de Projetos com Maximização de Valor Presente Líquido

Isac Mendes Lacerda

Tese de Doutorado submetida ao Programa de Pós-graduação em Informática do Instituto de Matemática e do Instituo Tércio Pacitti da Universidade Federal do Rio de Janeiro - UFRJ, como parte dos requisitos necessários à obtenção do título de Doutor em Informática.

Aprovada em 04 de novembro de 2022.

BANCA EXAMINADORA:

Prof. Jayme Luiz Szwarcfiter, Ph.D., UFRJ (Presidente)

Participação por vídeo conferência

Prof. Eber Assis Schmitz, Ph.D., UFRJ (Coorientador)

Participação por vídeo conferência

Profª. Rosiane de Freitas Rodrigues, D.Sc., UFAM (Coorientadora)

Participação por vídeo conferência

Profª. Lilian Markenzon, Ph.D., UFRJ

Participação por vídeo conferência

Prof. Mitre Costa Dourado, Ph.D., UFRJ

Participação por vídeo conferência

Prof. Heitor Luiz Murat de Meirelles Quintella, Ph.D., UFF

Participação por vídeo conferência

Prof. Rodrigo Clemente Thom de Souza, Ph.D., UFPR

Dedico este trabalho à minha querida esposa Marina e ao meu amado filho Davi.

AGRADECIMENTOS

Agradeço a Deus pelas oportunidades, pela saúde para enfrentar as limitações e por colocar em meu caminho pessoas que me abençoam muito além do que mereço. Sobre este trabalho, entre as pessoas que me ajudaram de forma direta e indireta, destaco:

- Professor Jayme Szwarcfiter me recebeu no programa, acreditou em meu trabalho, viabilizou participação em vários eventos, ajudou a resolver muitos problemas administrativos, além das sessões teóricas/técnicas e conselhos cirúrgicos. Professor incansável e sempre disponível. Além disso, tratou-me com uma consideração e educação que eu nunca vou esquecer. Muito obrigado, Professor Jayme!

- Professor Eber Schmitz ministrou a maior parte das disciplinas que cursei. Fui contagiado pelas temáticas e instrumentos de trabalho que ele apresentou. Aceitou me coorientar e ajudar no delineamento do escopo da pesquisa junto aos Professores Jayme e Rosiane. O Professor Eber abriu várias clareiras que serviram de passagem para que eu pudesse caminhar. Ajudou-me a ter os pés no chão, corrigindo erros importantes. Sua polivalência técnica/teórica se tornou modelo pra mim. Muito obrigado, Professor Eber!

- Professora Rosiane de Freitas aceitou me coorientar e teve mais paciência comigo do que eu mesmo. Ajudou-me a enfrentar as limitações com honestidade. Deu-me conselhos preciosos e me pôs a pensar. Foram várias sessões teóricas/técnicas de análise dos algoritmos, muitas delas depois das 22h (horário em que eu terminava um dos meus trabalhos). Mostrou o caminho pra que eu pudesse participar de eventos tradicionais da área, como a conferência europeia e a escola latino americana de pesquisa operacional. Não vou me esquecer disso. Muito obrigado, Professora Rosiane!

- Professor Heitor Quintella foi meu orientador de mestrado (abriu a porta da academia), apresentou-me o Professor Jayme e aceitou avaliar meu trabalho. Muito obrigado mais uma vez, Professor Heitor!

- Professores Lilian Markenzon, Mitre Dourado e Rodrigo Clemente aceitaram fazer parte da minha banca e avaliar meu trabalho. Muito obrigado, Professores!

- Minha querida esposa Marina e meu amado filho Davi que fazem parte do time da minha vida. Olhando para o brilho dos olhos deles fui renovado a caminhar quando a vontade de interromper era grande demais. Muito obrigado, Marina e Davi!

- Minha mãe que me ensinou, pelo exemplo, que as dificuldades da vida devem ser enfrentadas com ação, não importa quais sejam. Muito obrigado, mãe!
- Professor Fernando Sérgio de Moraes, líder do Grupo de Inferência de Reservatório (GIR) da UENF. Durante boa parte do curso de doutorado trabalhei paralelamente no GIR. Nesse tempo, fui apoiado pelo projeto do GIR e encorajado pelo Professor Fernando a seguir em frente. Professor Fernando, muito obrigado!
- Professor Douglas Valiati, coordenador do curso de Sistemas de Informação da Faculdade Municipal de Macaé-RJ, obrigado pelo apoio e incentivo!
- Evandro Ribeiro, amigo pessoal e profissional. Em tempo de doutorado, estivemos juntos no projeto do GIR e na minha breve passagem pela empresa Tata Consultancy Services (TCS). Obrigado pelo apoio, meu amigo!
- Colegas da unidade de TIC no campus de Macaé-RJ da UFRJ, onde trabalho (no tempo da edição deste texto). Agradeço o apoio, especialmente de Adriano Souza (diretor da TIC), Patrick Helder, Adriano Gonçalves e Paulo Freitas. O apoio de vocês na reta final foi fundamental. Muito obrigado!

Eu plantei, Apolo regou; mas Deus deu o crescimento.
(Bíblia Sagrada, 1 Coríntios 3:6)

RESUMO

Esta pesquisa apresenta uma análise empírica do desempenho de três algoritmos de escalonamento de projetos que visam maximizar o valor presente líquido com recursos irrestritos. Os algoritmos selecionados são: *Recursive Search* (RS), *Steepest Ascent Approach* (SAA) e *Hybrid Search* (HS). A principal motivação para esta pesquisa se refere ao desconhecimento sobre as complexidades computacionais dos algoritmos referidos (RS, SAA e HS), uma vez que todos os estudos até o momento apresentam lacunas nas análises. Além disso, as análises empíricas realizadas e registradas na literatura até agora não consideram o fato de um dos algoritmos (HS) usar uma estratégia de busca dupla, o que melhora marcantemente o desempenho desse algoritmo, enquanto os outros não. Para obter uma comparação justa de desempenho, implementamos a estratégia de busca dupla nos outros dois algoritmos (RS e SAA). Com a implementação, os novos algoritmos foram chamados de *Recursive Search Forward-Backward* (RSFB) e *Steepest Ascent Approach Forward-Backward* (SAAFB). Assim, RSFB, SAAFB e HS foram submetidos a um experimento fatorial em três amostras com diferentes características de rede de projeto (14.000 instâncias ao todo). Os resultados foram analisados por meio de Modelos Lineares Generalizados (MLG), que mostraram: a) os custos computacionais gerais de RSFB ($O(n^3)$), SAAFB e HS ($O(n^2)$ ou $O(n^3)$); b) os custos de reinício de buscas nas respectivas árvores geradoras como parte do custo total dos algoritmos ($O(n)$); c) e diferenças estatisticamente significativas entre as distribuições dos resultados dos algoritmos.

Palavras-chave: Avaliação empírica; algoritmos de escalonamento de projetos; *max-npv*; recursos irrestritos.

ABSTRACT

This research presents an empirical performance analysis of three project scheduling algorithms dealing with maximizing projects' net present value with unrestricted resources. The selected algorithms are: *Recursive Search* (RS), *Steepest Ascent Approach* (SAA), and *Hybrid Search* (HS). The main motivation for this research is the lack of knowledge about the computational complexities of the mentioned algorithms since all studies to date show some gaps in the analysis. Furthermore, the empirical analysis performed to date does not consider the fact that one algorithm (HS) uses a dual search strategy, which markedly improved the algorithm's performance, while the others don't. In order to obtain a fair performance comparison, we implemented the dual search strategy into the other two algorithms (RS and SAA). With the implementation, the new algorithms were called *Recursive Search Forward-Backward* (RSFB), and *Steepest Ascent Approach Forward-Backward* (SAAFB). Thus, RSFB, SAAFB, and HS were submitted to a factorial experiment with three different project network sampling characteristics (14,000 instances in all). The results were analyzed using the Generalized Linear Models (GLM) statistical modeling technique that showed: a) the general computational costs of RSFB ($O(n^3)$), SAAFB, and HS ($O(n^2)$ or $O(n^3)$); b) the costs of restarting the search in the spanning tree as part of the total cost of the algorithms ($O(n)$); c) and statistically significant differences between the distributions of the algorithms' results.

Keywords: Empirical evaluation; *max-npv*; project scheduling algorithms; unrestricted resources.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto do tema, a situação problema, os objetivos da pesquisa, as hipóteses assumidas e descreve brevemente a abordagem metodológica.

1.1 CONTEXTO

O Valor Presente Líquido (VPL), provavelmente o método mais utilizado para avaliação financeira de projetos, consiste em calcular a soma de todos os fluxos de caixa descontados gerados pelas atividades de um projeto. A premissa por trás do método é que quanto maior o VPL mais atrativo financeiramente é o projeto. No entanto, a aplicação desse método pode envolver diversas restrições para a realização do projeto, tais como: a) precedência entre as atividades, com tipos diferentes; b) datas impostas para início ou término das atividades; c) atraso (*time-lag*) entre as atividades; e) quantidades e tipos de recursos mobilizados; f) capacidade de renovação de recursos, entre outras.

Desse modo, obter o VPL de um projeto (sujeito a restrições), trata-se de um problema de maximização com muitas soluções possíveis. Cada solução tem seu próprio agendamento de datas para as atividades. No contexto de problemas de otimização, agendamento também é chamado de escalonamento. No caso, escalonamento se refere não só ao agendamento de datas mas também à designação de recursos às atividades do projeto ao longo do tempo.

As primeiras discussões sistemáticas sobre como maximizar o VPL de projeto, também chamado por problema de *max-npv* (do inglês *maximization of net present value*), originaram-se no trabalho de Russell (1970). A contribuição desse autor abriu caminho para pesquisas de diferentes classes de problemas de *max-npv*, como mostram várias publicações posteriores. Esse histórico inclui a revisão de literatura feita por Herroelen, Dommelen e Demeulemeester (1997) que registrou trinta e quatro trabalhos relacionados ao assunto e os agrupou em seis categorias.

No caso, o trabalho de revisão referido inclui uma categoria denominada *Deterministic Unconstrained max-npv* com a seguinte caracterização: a) Durações determinísticas para as atividades; b) Relações de precedência do tipo *término-início* (*finish-start)* com atraso (*time-lag*) zero entre as atividades; c) Fluxos de caixa determinísticos e; d) Taxa de desconto determinística. Esse tipo de problema consiste no cálculo do conjunto de datas de início de todas as atividades que maximizam o VPL do projeto e se refere à categoria de interesse desta tese.

Embora sete trabalhos tenham sido catalogados na categoria chamada *Deterministic Unconstrained max-npv* da revisão de Herroelen, Dommelen e Demeulemeester (1997), três novos trabalhos publicados e relacionados foram identificados. Dois desses trabalhos propuseram novos algoritmos e o terceiro trabalho propôs um refinamento para um algoritmo previamente catalogado. Os algoritmos desses trabalhos são *Recursive Search* (RS), com aresta extra, de Vanhoucke, Demeulemeester e Herroelen (1999), *Steepest Ascent Approach* (SAA) de Schwindt e Zimmermann (2001) e *Hybrid Search* (HS) de Vanhoucke (2006).

Nesse contexto, a principal motivação para esta tese está relacionada ao desconhecimento sobre as complexidades computacionais dos algoritmos RS, SAA e HS, uma vez que todos apresentam lacunas em suas análises. Uma motivação adicional foi que as análises empíricas realizadas, até o presente momento, não consideraram que apenas um dos algoritmos (HS) utiliza uma estratégia que escolhe a direção da busca, com marcante capacidade de melhora de desempenho, enquanto os outros não. Portanto, para obter uma comparação justa de desempenho, implementamos a estratégia de escolha da direção da busca nos outros dois algoritmos (RS e SAA). Por isso, as novas versões desses algoritmos foram denominadas como *Recursive Search Forward-Backward* (RSFB) e *Steepest Ascent Approach Forward-Backward* (SAAFB).

Desse modo, esta tese apresenta os resultados de um estudo empírico com os desempenhos dos algoritmos RSFB, SAAFB e HS. Eles foram submetidos a um experimento fatorial em três amostras de redes de projeto com características diferentes (que totaliza 14.000 instâncias), utilizando duas métricas de desempenho. A primeira métrica que trata do custo total é um apontador (*proxy*) para a medida de complexidade na Notação O, enquanto a segunda métrica trata do número de buscas que os algoritmos fazem nas respectivas árvores geradoras.

1.2 SITUAÇÃO PROBLEMA

O que caracteriza a situação problema desta pesquisa são as duas motivações anunciadas na seção anterior, mas que devem ser enfatizadas: 1) Em primeiro lugar, os três algoritmos (RS, SAA e HS) apresentam ordem de complexidade global desconhecida, ou seja, questão aberta. Parte dessa questão (em cada um dos algoritmos) se deve ao fato do número de buscas nas respectivas árvores geradoras também se tratar de questão aberta; 2) Em segundo lugar, o uso exclusivo da estratégia de escolha da direção da busca que o algoritmo HS tem e os algoritmos RS e SAA não têm.

Assim, a situação problema pode ser expressa através de duas perguntas:

- **É possível realizar um experimento que evidencie a ordem de complexi-**

dade global (Notação O) dos algoritmos, considerando a incorporação da estratégia de escolha da direção da busca nos três algoritmos (RS, SAA e HS)?

- **É possível realizar um experimento que evidencie a ordem de complexidade do número de buscas nas respectivas árvores geradoras, considerando a incorporação da estratégia de escolha da direção da busca nos três algoritmos (RS, SAA e HS)?**

1.3 OBJETIVOS DA PESQUISA

A partir da situação problema foram delineados os objetivos abaixo.

1.3.1 Objetivo geral

O objetivo geral da pesquisa é identificar evidências para as lacunas de ordem de complexidade (Notação O), considerando a incorporação da estratégia de escolha de direção da busca nos três algoritmos (RS, SAA e HS).

1.3.2 Objetivos específicos

- Incorporar a estratégia de escolha da direção da busca nos algoritmos RS e SAA e medir os seus resultados. Com a estratégia incorporada, RS e SAA passam a ser denominados como RSFB e SAAFB (FB de *Forward-Backward*).
- Identificar evidências para as lacunas de ordem global de complexidade dos algoritmos.
- Identificar evidências para as lacunas de ordem de complexidade do número de buscas dos algoritmos nas respectivas árvores geradoras.

1.4 HIPÓTESES DA PESQUISA

A formulação da situação problema e o delineamento dos objetivos cooperaram para a elaboração de respostas provisórias passíveis de teste. Em outras palavras, essas respostas servem como solução provisória em formato de proposições testáveis. Tais proposições serão chamadas de hipóteses e seguem anunciadas.

- **Hipótese I:**
  O algoritmo RSFB **está** em ordem (classe) de complexidade global maior que os algoritmos SAAFB e HS.

- **Hipótese II:**
  Os algoritmos HS e SAAFB **estão** na mesma ordem (classe) de complexidade global.

- **Hipótese III:**
  Os algoritmos RSFB, SAAFB e HS **estão** na mesma ordem (classe) de complexidade, no que se refere ao número de buscas nas respectivas árvores geradoras.

1.5 PRÉVIA METODOLÓGICA

A abordagem metodológica desta tese toma o método hipotético-dedutivo de Popper (2004) como referência. Todos os aspectos da aplicação dessa abordagem estão detalhados no Capítulo 5 que trata da metodologia. No entanto, cabe breve antecipação. Assim, Popper (2004) afirma que a existência de lacunas de conhecimento em teorias previamente estabelecidas são significativas fontes de pesquisa. Tais lacunas podem ser formuladas como situação problema, servindo como referência para a confecção de respostas provisórias (hipóteses) de natureza testável. As hipóteses funcionam como guia no desenvolvimento da pesquisa e, ao serem testadas, promoverão um novo estágio de conhecimento científico (com as hipóteses corroboradas ou refutadas).

2 TRABALHOS RELACIONADOS

Este capítulo apresenta uma síntese dos trabalhos da literatura relacionados ao problema de escalonamento de interesse, enfatizando as principais características de cada um deles.

2.1 SÍNTESE DOS TRABALHOS

A revisão da literatura de Herroelen, Dommelen e Demeulemeester (1997), intitulada de *Project network models with discounted cash flows a guided tour through recent developments*, descreve alguns dos trabalhos relacionados a esta tese. Tal revisão registrou sete trabalhos em uma categoria denominada *Deterministic Unconstrained max-npv*, com relação direta ao problema de escalonamento de interesse desta pesquisa. No entanto, foram identificados mais três novos trabalhos relacionados ao problema de interesse publicados depois da referida revisão. Dessa maneira, as principais características desses dez trabalhos estão descritas na sequência.

Tabela 1 – Síntese dos trabalhos relacionados.

Autores	**Algoritmo**	**Complexidade**	**Experimento**
1. Russell (1970)	sim	não	não
2. Grinold (1972)	sim	não	não
3. Smith-Daniels (1986)	não	não	não
4. Elmaghraby e Herroelen (1990)	sim	não	não
5. Herroelen e Gallens (1993)	sim	não	sim
6. Kazaz e Sepil (1996)	sim	não	sim
7. Demeulemeester, Herroelen e Dommelen (1996)	sim	não	sim
8. Vanhoucke, Demeulemeester e Herroelen (1999)	sim	parcial	sim
9. Schwindt e Zimmermann (2001)	sim	parcial	sim
10. Vanhoucke (2006)	sim	não	sim

A Tabela 1 apresenta um resumo desses trabalhos relacionados, onde **Algoritmo** significa que o trabalho descreve um algoritmo, **Complexidade** que apresenta alguma análise de complexidade e **Experimento** se o trabalho detalha alguma forma de análise empírica. Já a Figura 1 destaca a cronologia por década da produção dos trabalhos relacionados.

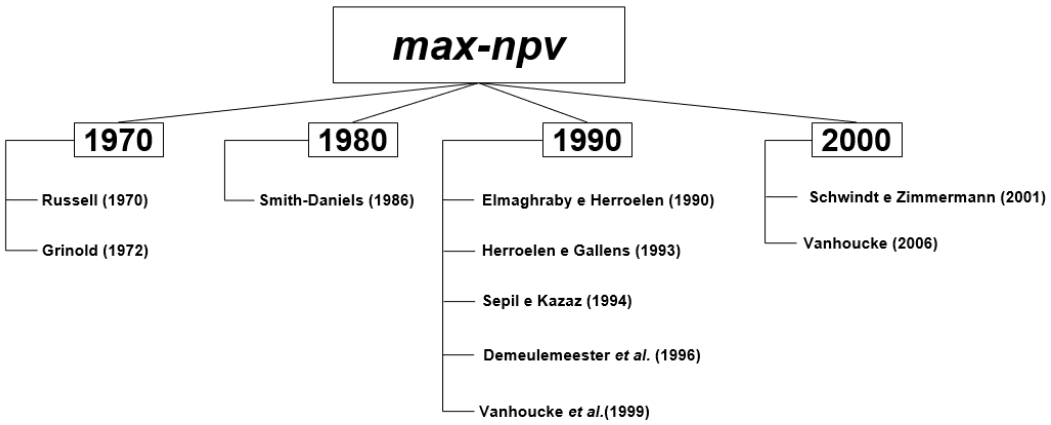

Figura 1 – Cronologia dos trabalhos relacionados.

2.2 TRABALHO 1: Russell (1970)

O primeiro trabalho, intitulado como *Cash flows in networks*, foi publicado por Russell (1970), um dos pioneiros em promover e formular a maximização do valor presente líquido em projetos. Nesse trabalho, o autor mostrou que, embora o problema envolva maximizar uma função objetivo não linear (sujeita a restrições lineares), é possível obter uma solução com sucessivas aplicações de programação linear. O autor propôs ainda um algoritmo composto por várias etapas, mas não apresentou nenhuma discussão sobre sua complexidade ou avaliação empírica.

2.3 TRABALHO 2: Grinold (1972)

No segundo trabalho, intitulado de *The payment scheduling problem*, Grinold (1972) também se preocupou em demonstrar que é possível ter uma solução para a função objetivo não linear (com restrições lineares) usando programação linear. O autor mostrou que a melhor solução para o problema deve ser obtida em uma árvore (viável) extraída da rede do projeto e que a busca pode ser restrita a estruturas em árvore. Desse modo, o autor propôs dois algoritmos: o primeiro resolve o problema por meio de um único *deadline* e o segundo resolve o problema por meio de vários *deadlines*. Esses algoritmos estão relacionados ao chamado procedimento especial de Markowitz para o problema de distribuição ponderada com o uso de sistemas triangulares de equações. Esse trabalho não apresentou experimento.

2.4 TRABALHO 3: Smith-Daniels (1986)

No terceiro trabalho, intitulado de *Summary measures for predicting the net present value of a project*, Smith-Daniels (1986) propôs um modelo de previsão para *max-npv* com recursos irrestritos, considerando um único fluxo de caixa na conclusão do projeto, o que é característico em projetos sob contratos de preço fixo. Além disso, medidas de resumo foram propostas para prever o valor de *max-npv*. No entanto, o trabalho não propôs algoritmos ou experimentos computacionais.

2.5 TRABALHO 4: Elmaghraby e Herroelen (1990)

No quarto trabalho, intitulado como *The scheduling of activities to maximize the net present value of projects*, Elmaghraby e Herroelen (1990) demonstraram que as abordagens dos artigos de Russell (1970) e Grinold (1972) podem produzir resultados inconclusivos, pois não especificam explicitamente restrições de início e fim para o escalonamento. Sob esses argumentos, Elmaghraby e Herroelen (1990) propuseram um algoritmo que constrói estruturas do tipo árvore de forma iterativa e usa determinação de intervalos de deslocamento. No entanto, o artigo não apresentou discussão sobre complexidade nem experimentos computacionais.

2.6 TRABALHO 5: Herroelen e Gallens (1993)

No quinto trabalho, intitulado de *Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects*, Herroelen e Gallens (1993) apresentaram uma versão simplificada do algoritmo de Elmaghraby e Herroelen (1990). Os autores submeteram esse algoritmo a um experimento computacional que permitiu comparações empíricas com as soluções obtidas via programação linear usando o *software Super LINDO*. No entanto, o artigo não apresentou discussão sobre complexidade.

2.7 TRABALHO 6: Kazaz e Sepil (1996)

No sexto trabalho, intitulado como *Project scheduling with discounted cash flows and progress payments*, Kazaz e Sepil (1996) argumentaram que em projetos reais é mais comum que os fluxos de caixa sejam associados a períodos regulares (como meses) do que a eventos como a conclusão de atividades. Assim, eles propuseram que os custos das

atividades fossem divididos por suas durações e a formulação do problema por meio de programação inteira. Os autores também apresentaram regras de um gerador de redes aleatórias com resultados de um experimento através do *software LINDO*. No entanto, não apresentaram discussões sobre complexidade.

2.8 TRABALHO 7: Demeulemeester, Herroelen e Dommelen (1996)

No sétimo trabalho, intitulado por *An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem*, Demeulemeester, Herroelen e Dommelen (1996) descreveram um algoritmo chamado *Recursive Search* (RS) composto por três passos, que realiza buscas recursivas em estruturas em árvore, assumindo que o fluxo de caixa positivo deve ser antecipado o máximo possível e o fluxo de caixa negativo deve ser adiado o máximo possível. Esse trabalho apresentou um experimento computacional, mas não discutiu nenhum aspecto de complexidade.

2.9 TRABALHO 8: Vanhoucke, Demeulemeester e Herroelen (1999)

No oitavo trabalho, intitulado de *On maximizing the net present value of a project under resource constraints*, Vanhoucke, Demeulemeester e Herroelen (1999) propuseram um refinamento para o algoritmo RS através da adição de uma aresta extra na árvore geradora entre o último e o primeiro vértices, embora o foco principal do trabalho tenha sido *max-npv* com restrição de recursos. Essa aresta extra favoreceu o processo de recursão do algoritmo proposto inicialmente em Demeulemeester, Herroelen e Dommelen (1996). O trabalho de Vanhoucke, Demeulemeester e Herroelen (1999) também apresentou um experimento computacional, mas não discutiu a complexidade do algoritmo. No entanto, na versão de RS com a aresta extra, Demeulemeester e Herroelen (2006) e Vanhoucke, Demeulemeester e Herroelen (2000) afirmaram que cada um dos dois primeiros passos pode ser implementado com custo $O(n^2)$, mas a complexidade do terceiro passo foi dada como questão aberta.

2.10 TRABALHO 9: Schwindt e Zimmermann (2001)

No nono trabalho, intitulado de *A steepest ascent approach to maximizing the net present value of projects*, Schwindt e Zimmermann (2001) apresentaram um algoritmo denominado *Steepest Ascent Approach* (SAA) com uma abordagem generalizada para precedência de atividades, admitindo intervalos mínimos e máximos. O artigo apresentou um experimento computacional e destacou que os dois primeiros de seus três algoritmos

componentes podem ser implementados respectivamente com custo $O(n)$ e $O(m\ log\ m)$. No entanto, a complexidade do terceiro algoritmo componente foi considerada uma questão aberta.

2.11 TRABALHO 10: Vanhoucke (2006)

No décimo e último trabalho, intitulado por *An efficient hybrid search algorithm for various optimization problems*, Vanhoucke (2006) propôs um algoritmo chamado *Hybrid Search* (HS) que combinou estratégias de RS e SAA, além de incluir a capacidade de inverter a direção de busca e deslocamento quando o projeto tem mais da metade das atividades com fluxo de caixa negativo. Nesse trabalho, o autor apresentou os resultados de um experimento computacional, mas não discutiu nenhum aspecto de complexidade.

3 REFERENCIAL TEÓRICO

Este capítulo descreve as características gerais de escalonamento de projetos, a notação de 3-campos, as origens do método do VPL, o problema de escalonamento de interesse da tese, os algoritmos fundamentais considerados (*Recursive Search*, *Steepest Ascent Approach* e *Hybrid Search*) e exemplifica as estratégias desses algoritmos através de uma instância comum.

3.1 ESCALONAMENTO DE PROJETOS

Escalonamento se refere a uma área teórica com larga variedade de problemas que podem ser modelados matematicamente. Também é uma área prática que desperta interesse em diferentes setores produtivos com a preocupação de fazer uso eficiente de recursos (normalmente escassos). De modo geral, os problemas formulados de escalonamento levam em conta a alocação de atividades (trabalho a ser realizado) a máquinas (processadores das atividades) com a finalidade de otimizar algum(ns) aspecto(s) referente(s) aos recursos ao longo do tempo (BRUCKER, 2006; FREITAS, 2009; PINEDO, 2012).

Entre os contextos de aplicação de técnicas de escalonamento está o gerenciamento de projetos. Segundo o *Project Management Institute* (PMI, 2017), projeto se refere a um esforço temporário empreendido na criação de produto, serviço ou resultado único. Por sua vez, o gerenciamento de um projeto se refere a aplicação de conhecimentos, habilidades, ferramentas e técnicas às atividades desse projeto, com o propósito de alcançar seus objetivos. No caso, como um projeto tem natureza temporária e normalmente conta com recursos escassos é natural que sejam utilizadas técnicas de escalonamento em seu gerenciamento.

3.1.1 Estrutura Analítica de Projeto (EAP)

Parte dos objetivos de um projeto se refere ao cumprimento do seu escopo, ou seja, a realização das atividades que levam à conclusão dos chamados pacotes de trabalho, entregáveis do projeto (PMI, 2019). Esses pacotes se referem a uma decomposição do escopo normalmente arranjado hierarquicamente através da denominada Estrutura Analítica de Projeto (EAP). Embora uma EAP seja utilizada como referência para a identificação de atividades necessárias para a conclusão dos pacotes de trabalho, essa estrutura tem foco nos pacotes de trabalho (sem explicitar as atividades). Além disso, os pacotes de trabalho em uma EAP são os elementos de último nível na hierarquia, conforme mostra a Figura 2

com os elementos destacados em fundo preto. Todos os outros elementos (que não estão no último nível da hierarquia) são agrupadores de conveniência por algum critério de afinidade. É válido destacar que a EAP da Figura 2 se refere a alguns possíveis pacotes de trabalho em um projeto de um *software* genérico e hipotético. Portanto, tal figura não tem propósito de destacar todos os pacotes necessários de um projeto real desse tipo, além de não indicar uma abordagem específica de desenvolvimento.

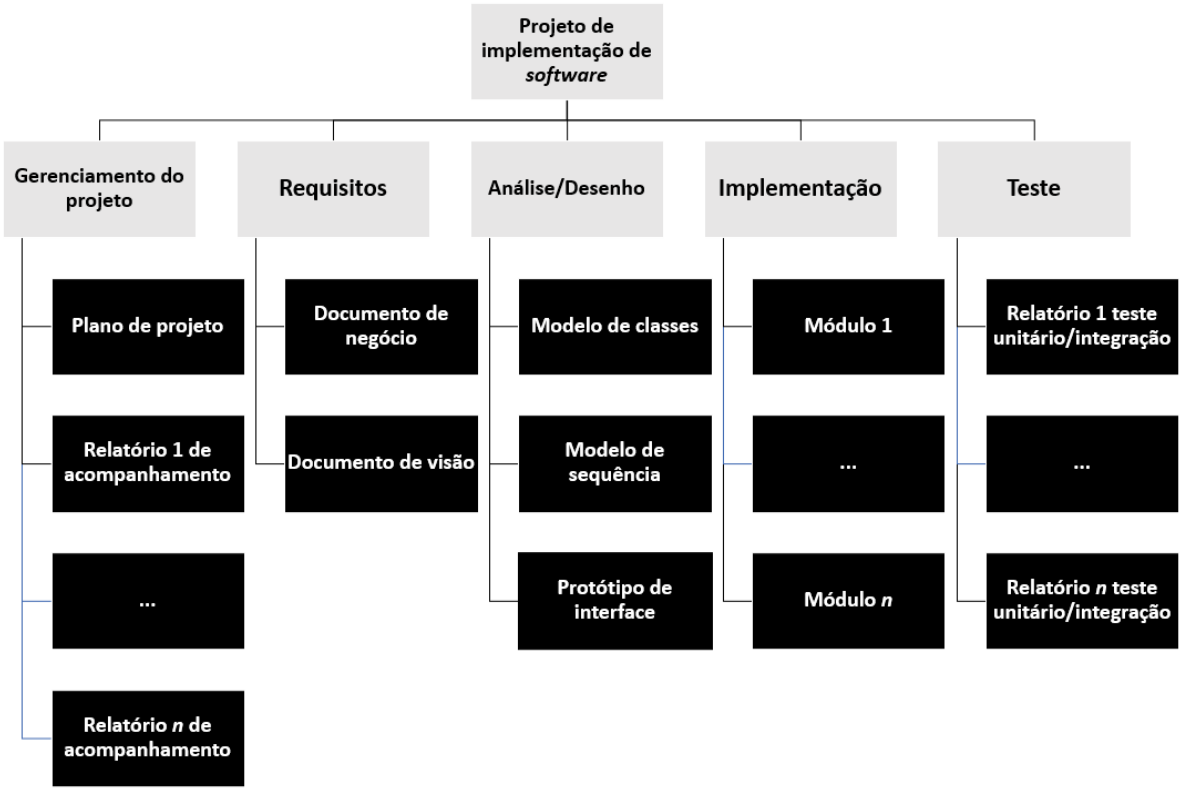

Figura 2 – Estrutura Analítica de Projeto (EAP) de um *software* genérico.

3.1.2 Atividades em Redes

Cada pacote de trabalho em uma EAP é decomposto em atividades que podem manter precedência entre si. Definir essas atividades e a sequência entre elas faz parte do escalonamento do projeto (também chamado de cronograma). Quatro tipos de relações de precedência, com possibilidade de atraso (*time-lag*) entre elas, são comumente utilizados, *Finish-Start* (FS), *Finish-Finish* (FF), *Start-Start* (SS) e *Start-Finish* (SF). Com isso, as atividades de projeto podem ser representas em estruturas em rede, como em um gráfico de Gantt ou em um grafo com notação *Activity-on-Node* (AoN). Como mostra a Figura 3, o gráfico de Gantt, com mais de cem anos de aplicação (WILSON, 2003; PINEDO, 2012), relaciona as atividades (expressas com barras horizontais) com setas, onde a unidade de medida no eixo horizontal se refere a tempo (*T1, T2, .., Tn*).

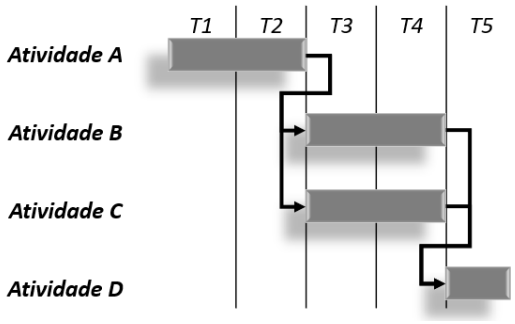

Figura 3 – Exemplo de Gráfico de Gantt.

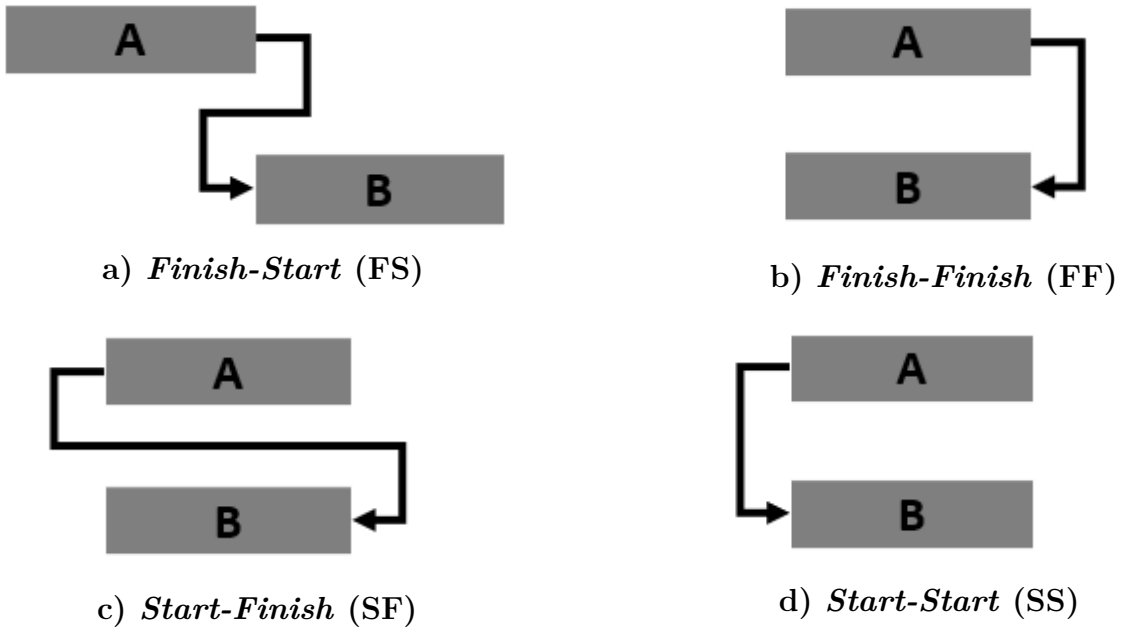

Figura 4 – Tipos de precedência.

Na Figura 4(a), a relação de precedência FS indica que a atividade B só pode ter início quando a atividade A for concluída. Na Figura 4(b), a relação de precedência FF indica que a atividade B só pode terminar se a atividade A tiver terminado antes. Na Figura 4(c), a relação de precedência SF indica que a atividade B só pode terminar se a atividade A tiver sido iniciada. Por último, na Figura 4(d), a relação de precedência SS indica que a atividade B só pode iniciar se a A tiver sido iniciada.

Na notação AoN, as atividades de um projeto são representadas por vértices e as precedências são representadas pelas arestas. A Figura 5 mostra um projeto com essa notação, no entanto, sem especificar *time-lag* e os tipos de precedência.

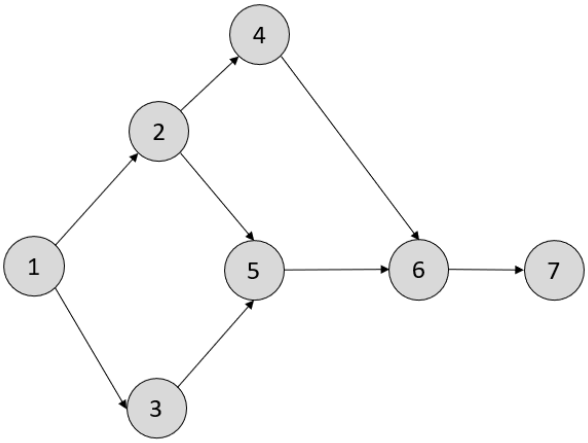

Figura 5 – Notação AoN.

Sobre *time-lag* é possível representá-lo de modo generalizado com valor mínimo ou máximo, de acordo com o tipo de relacionamento. A partir da notação de Elmaghraby e Kamburowski (1992) o *time-lag* mínimo pode ser expresso como: $FS_{ij}^{min}(x)$, $SF_{ij}^{min}(x)$, $SS_{ij}^{min}(x)$ ou $FF_{ij}^{min}(x)$. Nos casos de *time-lag* mínimo, a atividade j só pode iniciar (ou terminar) quando a atividade predecessora i já tiver iniciado (ou terminado) para um período de tempo expresso por x unidades. De modo análogo, o *time-lag* máximo pode ser expresso como: $FS_{ij}^{max}(x)$, $SF_{ij}^{max}(x)$, $SS_{ij}^{max}(x)$ ou $FF_{ij}^{max}(x)$. Nos casos de *time-lag* máximo, a atividade j deve iniciar (ou terminar) no máximo x períodos de tempo depois do início (ou término) da atividade i. Tal representação de *time-lag* pode ser empregada junto à notação AoN, como mostra a Figura 6.

a) *time-lag max* **b) *time-lag min***

Figura 6 – Exemplos de *time-lag*.

3.2 NOTAÇÃO DE 3-CAMPOS EM PROJETOS

Com grande variedade de possíveis problemas de escalonamento, a literatura tem se desenvolvido com o uso de uma representação conhecida como notação de 3-campos. Essa notação originalmente proposta por Graham et al. (1979) recebeu algumas extensões. Entre as extensões estão a de Blazewicz, Lenstra e Kan (1983) e a de Herroelen, Demeulemeester e Reyck (1997). No caso, o interesse desta tese está voltado para a extensão proposta em Herroelen, Demeulemeester e Reyck (1997), pois se refere a um esquema de classificação

especializado em escalonamento de projetos, levando em consideração particularidades do contexto de gerenciamento de projetos.

Na abordagem especializada para projetos, semelhantemente à original, três campos fazem parte da composição, $\alpha|\beta|\gamma$. O primeiro campo α descreve características dos recursos do projeto. O segundo campo β descreve características das atividades do projeto. Já o terceiro campo γ descreve a(s) medida(s) de desempenho, ou simplesmente, a função (ou funções) objetivo de interesse do projeto.

3.2.1 Características dos recursos de projeto: campo α

O campo destinado à caracterização dos recursos (α) de projeto pode ser dividido em até três subcampos: α_1, α_2 e α_3.

- O subcampo α_1 descreve a **quantidade de tipos de recursos**, considerando o domínio $\alpha_1 \in \{^\circ, 1, m\}$, com os respectivos significados:
  - $\alpha_1 = \ ^\circ$: os recursos são irrestritos.
  - $\alpha_1 = 1$: apenas um tipo de recurso é considerado.
  - $\alpha_1 = m$: o número de tipos de recursos é m.
- O subcampo α_2 descreve o **tipo de recurso** utilizado, considerando o domínio $\alpha_2 \in \{^\circ,\ 1,\ T,\ 1T,\ v\}$, com os respectivos significados:
  - $\alpha_2 = \ ^\circ$: ausência de qualquer tipo de especificação de recursos.
  - $\alpha_2 = 1$: existência de recursos renováveis, cuja disponibilidade é especificada para um período de duração (ex. hora, turno, dia, semana, ou mês).
  - $\alpha_2 = T$: não existem recursos renováveis em todo o horizonte do projeto.
  - $\alpha_2 = 1T$: recursos renováveis e não renováveis, duplamente restritos, cuja disponibilidade é especificada tanto em um período específico quanto em todo horizonte do projeto.
  - $\alpha_2 = v$: recursos parcialmente (não) renováveis, cuja disponibilidade é renovada em períodos de tempo específicos.
- O subcampo α_3 descreve o **as características da disponibilidade dos recursos de projeto**, considerando o domínio $\alpha_2 \in \{^\circ,\ va\}$, com os respectivos significados:
  - $\alpha_3 = \ ^\circ$: os recursos renováveis estão disponíveis em quantidades constantes.
  - $\alpha_3 = va$: os recursos renováveis estão disponíveis em quantidades variáveis.

3.2.2 Características das atividades: campo β

O campo destinado à caracterização das atividades (β) de projeto pode ser dividido em até oito subcampos: β_1, β_2, β_3, β_4, β_5, β_6, β_7 e β_8.

- O subcampo β_1 descreve a possibilidade de **preempção**, considerando o domínio $\beta_1 \in \{^\circ,\ pmtn,\ pmtn\text{-}rep\}$, com os respectivos significados:
  - $\beta_1 = {}^\circ$: interrupções nas atividades não são permitidas.
  - $\beta_1 = pmtn$: atividades podem ser interrompidas e retomadas no ponto da interrupção.
  - $\beta_1 = pmtn\text{-}rep$: atividades podem ser interrompidas, mas não podem ser retomadas no ponto de interrupção, ou seja, devem ser completamente refeitas.

- O subcampo β_2 descreve as **restrições de precedência** entre as atividades, considerando o domínio $\beta_2 \in \{^\circ,\ cpm,\ min,\ gpr,\ prob\}$, com os respectivos significados:
  - $\beta_2 = {}^\circ$: atividades sem restrições de precedência.
  - $\beta_2 = cpm$: atividades sujeitas estritamente ao tipo de precedência *finish-start* com *time-lag* zero.
  - $\beta_2 = min$: as atividades estão sujeitas aos tipos de precedência *start-start, finish-start, start-finish* e *finish-finish* com *time-lag* mínimo.
  - $\beta_2 = gpr$: as atividades estão sujeitas a relações generalizadas de precedência *start-start, finish-start, start-finish* e *finish-finish* com *time-lag* mínimo e máximo.
  - $\beta_2 = prob$: rede de atividades é do tipo probabilística, onde a evolução do projeto correspondente não é determinada antecipadamente.

- O subcampo β_3 descreve os **tempos de prontidão** das atividades, considerando o domínio $\beta_3 \in \{^\circ,\ p_j\}$, com os respectivos significados:
  - $\beta_3 = {}^\circ$: atividades têm tempos de prontidão zero.
  - $\beta_3 = p_j$: tempo de prontidão diferente por atividade.

- O subcampo β_4 descreve as **durações** das atividades de projeto, considerando o domínio $\beta_4 \in \{^\circ,\ cont,\ d_j = d\}$, com os respectivos significados:
  - $\beta_4 = {}^\circ$: atividades têm durações inteiras arbitrárias.
  - $\beta_4 = cont$: atividades têm durações contínuas arbitrárias.

  - $\beta_4 = (d_j = d)$: todas as atividades têm duração igual a d unidades.

- O subcampo β_5 descreve possíveis *deadlines*, considerando o domínio $\beta_5 \in \{^\circ,\ \delta_j,\ \delta_n\}$, com os respectivos significados:

  - $\beta_5 =\ ^\circ$: não existem *deadlines*.
  - $\beta_5 = \delta_j$: existem *deadlines* impostos às atividades.
  - $\beta_5 = \delta_n$: existe *deadline* imposto ao projeto.

- O subcampo β_6 descreve a **natureza dos recursos requeridos** pelas atividades do projeto, considerando o domínio $\beta_6 \in \{^\circ,\ vr,\ disc,\ cont,\ int\}$, com os seguintes significados:

  - $\beta_6 =\ ^\circ$: atividades exigem os recursos em uma quantidade discreta e constante.
  - $\beta_6 = vr$: atividades requerem os recursos em quantidades discretas e variáveis.
  - $\beta_6 = disc$: os requisitos de recursos da atividade são uma função discreta da duração da atividade.
  - $\beta_6 = cont$: os requisitos de recursos da atividade são uma função contínua da duração da atividade.
  - $\beta_6 = int$: os requisitos de recursos da atividade são expressos como uma função de intensidade ou taxa.

- O subcampo β_7 descreve os **modos de execução** das atividades, considerando o domínio $\beta_7 \in \{^\circ,\ mu,\ id\}$, com os seguintes significados:

  - $\beta_7 =\ ^\circ$: as atividades devem ser executadas de um único modo.
  - $\beta_7 = mu$: as atividades têm múltiplos modos pré-especificados de execução.
  - $\beta_7 = id$: as atividades estão sujeitas a restrições de identidade de modo (restrições específicas por por grupos de atividades).

- O subcampo β_8 descreve **implicações financeiras** das atividades, considerando o domínio $\beta_8 \in \left\{^\circ,\ c_j,\ c_j^+,\ per,\ sched\right\}$, com os seguintes significados:

  - $\beta_8 =\ ^\circ$: não existe fluxo de caixa especificado para as atividades no problema de escalonamento.
  - $\beta_8 = c_j$: existe fluxo de caixa arbitrário para as atividades no problema de escalonamento.
  - $\beta_8 = c_j^+$: existe fluxo de caixa arbitrário e positivo para as atividades no problema de escalonamento.

- $\beta_8 = per$: fluxo de caixa periódico especificado para o projeto (ex.: pagamentos em intervalos regulares).
- $\beta_8 = sched$: tanto o valor quanto o momento dos fluxos de caixa devem ser determinados.

3.2.3 Características da(s) função(ões) objetivo: campo γ

O último campo desta notação (γ) é destinado às medidas de desempenho (ou critério de otimalidade). Os dois tipos principais de medida de desempenho são medidas de conclusão antecipada (também chamadas de regulares) e medidas de conclusão livre (também chamadas de não-regulares). Como existe uma infinidade de possíveis funções objetivo, seguem apenas alguns exemplos listados.

- $\gamma = C_{max}$: para minimização do *makespan* do projeto.
- $\gamma = T_{max}$: para minimização do atraso do projeto.
- $\gamma = early/tardy$: para minimização do peso da antecipação-atraso do projeto.
- $\gamma = max\text{-}npv$: para maximização do valor presente líquindo do projeto.

3.3 ORIGENS DO VPL

As propostas teóricas do início do século XX sobre capital, investimento e taxa de juros do notável economista Irving Fisher marcam o gênesis da literatura de ferramentas de apoio à tomada de decisão em projetos, considerando o critério financeiro (caso do método de Valor Presente Liquido). Nesse sentido, três dos trabalhos de Fisher podem ser apontados: no primeiro, o autor discute aspectos da natureza do capital e da renda (FISHER, 1906); no segundo, o autor apresenta aspectos da aplicação da taxa de juros de interesse sobre um empreendimento (projeto) (FISHER; BARBER, 1907); e no terceiro trabalho, o autor faz uma compilação que inclui as discussões dos dois primeiros trabalhos, além de teorias sobre orçamento de capital, mercado de títulos e taxas de inflação (FISHER, 1930).

A grande quantidade de livros textos (especialmente na literatura de negócios) que apresenta o método do Valor Presente Líquido (VPL) como ferramenta de apoio à tomada de decisão em projetos evidencia sua utilidade. A natureza desse método revela a preocupação que o tomador de decisão deve ter com a variação do poder de compra que o dinheiro tem ao longo do tempo. Tal natureza, está diretamente relacionada aos

postulados de Irving Fisher que incluem a dimensão do tempo no poder de compra do dinheiro.

Segundo Gitman, Juchau e Flanagan (2015) o método do VPL é uma técnica sofisticada para orçamento de capital que explicita o valor do dinheiro no tempo, apoiando à tomada de decisão. A abordagem de cálculo do VPL em um projeto envolve a subtração do investimento inicial (FC_0: fluxo de caixa no tempo 0) do somatório das entradas e saídas (FC_t: fluxo de caixa no tempo futuro t), descontadas a uma taxa de interesse (r). Desse modo, usar VPL significa considerar entradas e saídas de tempo futuro, mas medidas em tempo presente. Assim, uma primeira representação possível para a abordagem de cálculo está na expressão 3.1. Nessa expressão, além dos termos já mencionados, t se refere ao final de um período de tempo e n ao final do último período de tempo do projeto. Considerando que o investimento inicial (FC_0) sempre é negativo e ocorre no tempo 0, a expressão 3.1 pode ser simplificada, o que equivale à expressão 3.2.

$$VPL = \left(\sum_{t=1}^{n} \frac{FC_t}{(1+r)^t} \right) - FC_0 \tag{3.1}$$

$$VPL = \sum_{t=0}^{n} \frac{FC_t}{(1+r)^t} \tag{3.2}$$

Os critérios usuais para a seleção ou rejeição de projetos com apoio do método do VPL são os seguintes:

- Projeto $XPTO$ com VPL > 0 **é considerado viável**, pois suas receitas superam seus custos, com a taxa de interesse descontada.
- Projeto $XPTO_2$ com VPL ≤ 0 **é considerado não viável**, pois suas receitas não superam seus custos, com a taxa de interesse descontada.
- Se VPL do projeto $XPTO >$ VPL do projeto $XPTO_2$ (com capacidade de realização de um único projeto no mesmo período), **o projeto $XPTO$ deve ser selecionado**.

Para uma visualização com números, foram tomados emprestados dois exemplos de projetos apresentados por Gitman, Juchau e Flanagan (2015), conforme Figuras 7 e 8. Ambos os projetos têm fluxos de caixa expressos em um horizonte de cinco anos e taxa de desconto de 10% ($r = 10\%$).

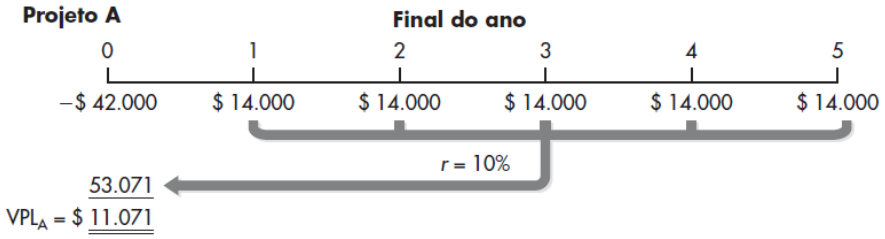

Figura 7 – VPL projeto A.

No projeto A (Figura 7), todas as entradas positivas ao final dos anos 1, 2, 3, 4 e 5 seguem destacadas (barra cinza mais espessa). Considerando os descontos individuais (de 10% por entrada), a soma das entradas é igual a \$53.071 dinheiros. Ao subtrair o fluxo de caixa inicial (FC_0), o VPL do projeto é dado como \$11.071 dinheiros.

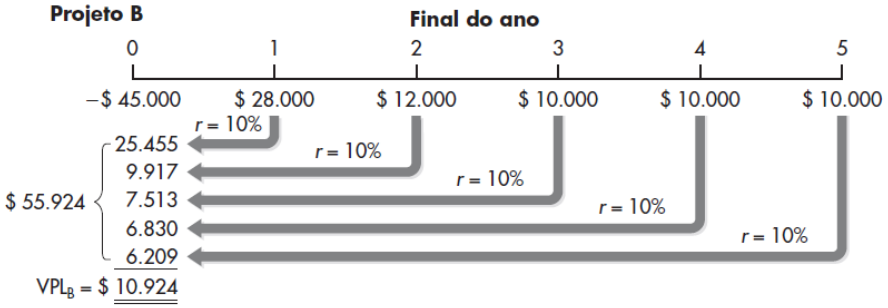

Figura 8 – VPL projeto B.

No projeto B (Figura 8), as entradas positivas, dessa vez com valores variados, também seguem destacas ao final dos anos 1, 2, 3, 4 e 5. Diferente do primeiro exemplo, no projeto B, o desconto de cada entrada segue detalhado. A soma das entradas (com os devidos descontos de 10%) é igual a \$55.924 dinheiros. Assim, ao subtrair o fluxo de caixa inicial (FC_0), o VPL do projeto B é dado por \$10.924 dinheiros. Embora os projetos A e B sejam viáveis, pelo critério exclusivamente do VPL, o projeto A é considerado melhor que o B.

Para fechar esta seção, deve ser dito que embora as raízes do cálculo básico do método de VPL estejam nos trabalhos de Irving Fisher, a discussão pioneira sobre maximização de VPL, considerando várias possíveis restrições, surgiu apenas na década de 1970 com o trabalho de Russell (1970). Em outras palavras, a formulação da maximização do VPL em projetos como problema de otimização surgiu na literatura apenas nos anos 70, o que abriu caminho para a formulação de várias classes de problemas, assim como propostas de algoritmos.

3.4 O PROBLEMA DE INTERESSE

O problema de interesse desta tese, para maximização de VPL, refere-se a projetos com restrição de precedência estritamente do tipo *finish-start* entre as atividades, com *lag* zero entre elas e recursos irrestritos. Nesse sentido, as informações sobre as variáveis e restrições do problema podem ser apresentadas com o apoio de um grafo denominado $G(V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas. Nesse grafo, dois vértices especiais são chamados *dummy* inicial e *dummy* final, respectivamente demarcando o início e o fim do projeto. Cada vértice (representando uma atividade do projeto) está associado ao seguinte conjunto de variáveis (atributos): a) fluxo de caixa não descontado (no caso dos *dummies* com valor zero); b) uma duração; c) uma data de início; d) uma data final. Os vértices são organizados em ordem crescente, começando no *dummy* inicial e terminando no *dummy* final. As arestas indicam a relação de precedência entre atividades. Além disso, a data final do último *dummy* deve ser menor ou igual ao *deadline* do projeto.

A Figura 9(a) é um exemplo da representação de grafo G de um projeto hipotético, com alguns dos atributos mencionados. Nesse exemplo, os números indicados sobre os vértices significam as durações das atividades, os números indicados abaixo dos vértices significam os fluxos de caixa não descontados das atividades, as setas indicam as precedências, os vértices 1 e 8 (em fundo branco) indicam os *dummies*, os vértices 2 e 3 (em fundo preto) destacam as atividades com fluxo de caixa negativo e os vértices 4, 5, 6 e 7 (em fundo cinza) destacam as atividades com fluxo de caixa positivo. A Figura 9(b) se refere ao mesmo grafo da Figura 9(a), mas acrescida de uma aresta extra entre os *dummies* com um *lag* negativo que expressa o *deadline* do projeto. Essa aresta extra é utilizada por alguns dos algoritmos (descritos na próxima seção) no processo de busca por atividades a deslocar.

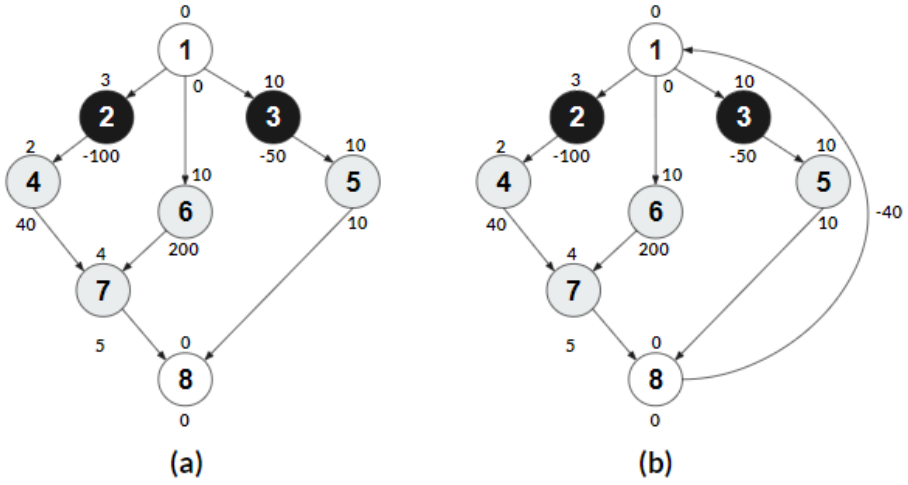

Figura 9 – Grafo G (sem e com aresta extra).

Usando a notação de 3-campos, especializada para o contexto de projetos por Herroelen, Demeulemeester e Reyck (1997), o problema de interesse pode ser expresso como: $\circ \mid cpm, \delta_n, c_j \mid max\text{-}npv$. Com essa notação, $\circ$ indica uso de recursos irrestritos, *cpm* indica precedência entre atividades *finish-start* com *lag* zero; δ_n indica que existe um *deadline* para o projeto; c_j denota um fluxo de caixa para cada atividade; e *max-npv* significa que a função objetivo é maximizar o valor presente líquido (onde *npv* é o acrônimo para os termos em inglês *net present value*). Assim, a função objetivo pode ser dada como:

$$max \sum_{i=2}^{n-1} c_i.e^{-\alpha(s_i+d_i)} \tag{3.3}$$

Nesse caso, c_i indica o fluxo de caixa de cada atividade i (no seu tempo de término), s_i indica o início de cada atividade, d_i indica a duração de cada atividade e $e^{-\alpha(s_i+d_i)} = 1/(1+r)^{(s_i+d_i)}$ indica o fator de desconto, onde r se refere à taxa de desconto. Essa expressão está sujeita às seguintes restrições:

$$s_i + d_i \leq s_j;\ \forall (i,j) \in E \tag{3.4}$$

$$s_1 = 0 \tag{3.5}$$

$$s_n \leq \delta_n \tag{3.6}$$

$$s_i \in \mathbb{N}; i = 2, 3, 4..., n \tag{3.7}$$

A expressão 3.3 indica que a função objetivo é maximizar o VPL. A expressão 3.4 limita o término de qualquer atividade para que seja menor ou igual ao início de qualquer uma de suas atividades sucessoras (s_j). A expressão 3.5 indica que a data de início do *dummy* inicial seja igual a zero. A expressão 3.6 indica que o início do *dummy* final (s_n) deve ser menor ou igual ao *deadline* do projeto (δ_n). A última expressão 3.7 indica que as datas de início das atividades devem pertencer ao conjunto dos números naturais.

Vale ressaltar que alguns autores já mostraram que, para esta classe de problema, é possível obter uma solução em tempo polinomial (CAVALCANTE et al., 2001; CHANG; EDMONDS, 1985; GRÖFLIN; LIEBLING; PRODON, 1982; JUANG, 1994; ROUNDY et al., 1991; SANKARAN; BRICKER; JUANG, 1999). Além disso, Grinold (1972) demonstrou que é possível transformar o problema de programação não linear em um problema de programação linear usando uma abordagem orientada a eventos.

Na seção seguinte são descritos os algoritmos considerados para o problema anunciado. Tais algoritmos utilizam tipos diferentes de árvores geradoras como parte de suas respectivas estratégias de solução. Os grafos da Figura 10 destacam os dois tipos de árvores geradoras utilizadas. Na Figura 10(a) dois pontos importantes devem ser observados: a) as arestas tracejadas foram eliminadas da árvore geradora, mas continuam existindo no grafo

G; b) uma aresta extra entre os *dummies* foi inserida. Na Figura 10(b) deve ser observado que as arestas tracejadas também foram removidas e o *dummy* final não está conectado diretamente ao *dummy* inicial.

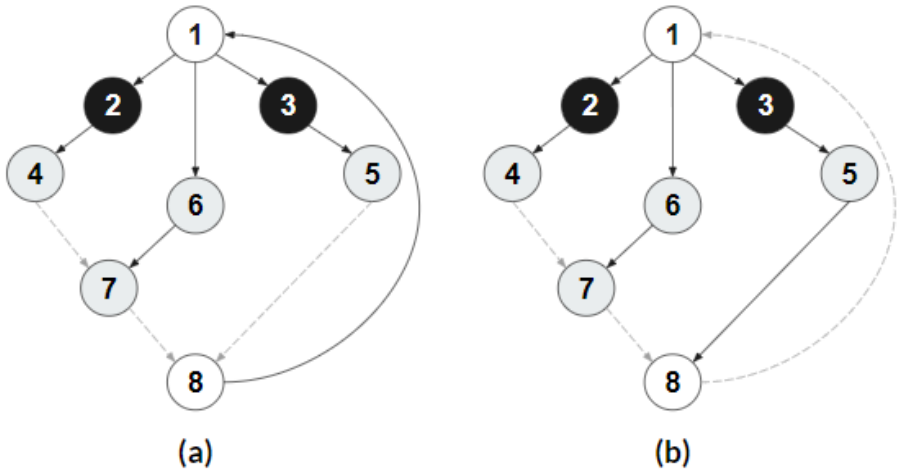

Figura 10 – Árvore geradora (com e sem aresta extra).

3.5 ALGORITMOS PARA O PROBLEMA DE INTERESSE

Os algoritmos fundamentais desta tese (para o problema $\circ \mid cpm, \delta_n, c_j \mid max\text{-}npv$) são *Recursive Search* (RS), *Steepest Ascent Approach* (SAA) e *Hybrid Search* (HS). Sobre eles, esta seção apresenta: a) a descrição de suas respectivas estratégias; b) os pseudocódigos correspondentes aos seus componentes (conforme as versões originais); c) e um exemplo comum de projeto sendo escalonado cena por cena, de modo a reforçar as respectivas estratégias. Nesse sentido, a Figura 11 apresenta uma síntese com cada um dos algoritmos e seus componentes.

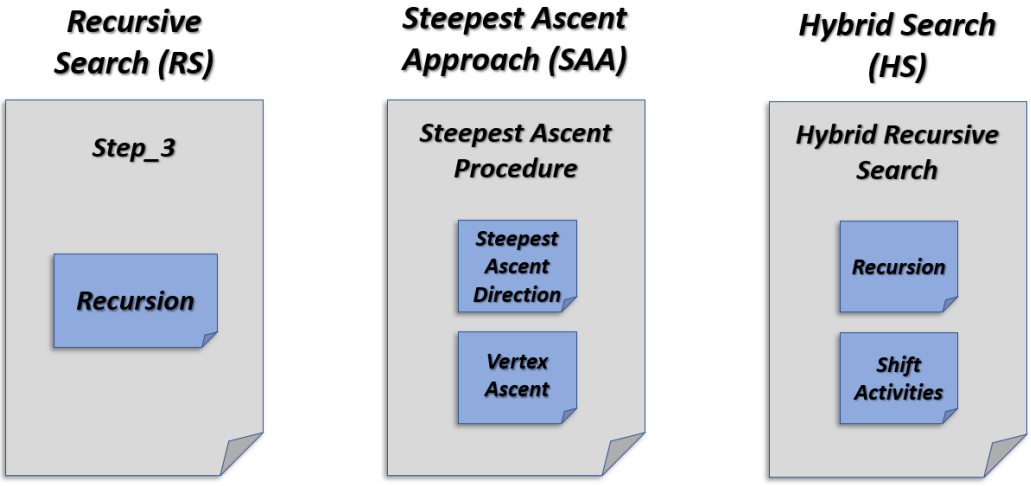

Figura 11 – Algoritmos para $\circ \mid cpm, \delta_n, c_j \mid max\text{-}npv$.

3.5.1 *Recursive Search*

Recursive Search (RS) é um dos algoritmos fundamentais nesta pesquisa. Esse algoritmo foi originalmente proposto por Demeulemeester, Herroelen e Dommelen (1996) e refinado por Vanhoucke, Demeulemeester e Herroelen (1999). A versão refinada, também apresentada no trabalho de Demeulemeester e Herroelen (2006), é a versão considerada neste texto. A estrutura de RS inclui três passos. No primeiro passo, RS cria uma árvore denominada *Early Tree* (ET), com datas antecipadas quanto possível para os vértices (atividades). RS assume a existência de uma aresta entre o primeiro vértice e último vértice (*dummies*) na ET, útil ao processo de recursão. No segundo passo, RS cria uma árvore denominada *Current Tree* (CT), cópia da ET. Ainda no segundo passo, vértices sem sucessores com fluxo de caixa negativo são deslocados quanto possível, levando em consideração as restrições de precedência. No terceiro passo, RS busca recursivamente por subárvores com fluxo de caixa negativo na CT e cada uma delas é deslocada conforme as restrições.

Como os dois primeiros passos fazem apenas pré-processamento, todo o trabalho algorítmico interessante é feito no terceiro passo de RS que inclui dois algoritmos componentes: *Step_3* e *Recursion*. O algoritmo componente *Step_3* é utilizado para iniciar uma busca em profundidade recursiva, através da chamada ao segundo algoritmo componente *Recursion* (que busca por subárvores com fluxo de caixa negativo). Cada subárvore identificada através de *Recursion* é imediatamente deslocada e uma nova busca é iniciada a partir do primeiro vértice (*dummy* inicial) da CT, através de uma nova chamada ao algoritmo componente *Step_3*. Sobre a complexidade de RS, os autores afirmam que os dois primeiros passos podem ser implementados com custo $O(n^2)$, mas apontam o terceiro como questão aberta (VANHOUCKE; DEMEULEMEESTER; HERROELEN, 1999; DEMEULEMEESTER; HERROELEN, 2006). Um dos pontos destacados para tal questão se refere ao desconhecimento do número de vezes que as buscas são reiniciadas com *Step_3* na CT. **Como consequência, a complexidade global de RS é tida como questão aberta**.

O pseudocódigo de *Step_3* segue indicado em Algoritmo 1. Entre os termos que merecem destaque, CA se refere ao conjunto de atividades visitadas (*Considered Activities*), SA' se refere ao conjunto de atividades candidatas a deslocamento (*Set of Activities*) e DC' se refere ao fluxo de caixa descontado (*Discounted Cash Flow*) de SA'.

Algoritmo 1: *Step_3* - versão original.

```
1 procedure Step_3;
2     CA = ∅;
3     Do Recursion(1) → SA', DC'(parameters returned by the recursive function);
4     Report the optimal completion times of the activities and the net present value DC'. STOP.
```

Já o pseudocódigo de *Recursion* está indicado em Algoritmo 2. No caso, *Recursion* recebe como argumento um novo vértice (*newnode*) e assume que CA (*Considered Activities*) e CT (*Current Tree*) são estruturas globais. Na linha 9, deve ser destacada a função denominada *Compute* v_{k*l*} que calcula a menor distância entre todos os vértices de SA' em relação a vértices fora de SA', conforme as precedências do projeto. Desse modo, f_l indica o fim da atividade $l \notin SA'$, d_l indica a duração de l e f_k indica o fim da atividade $k \in SA'$. Além disso, $(k*, l*)$ se refere à aresta contida no conjunto A (com todas as restrições de precedência do projeto) que representa a menor distância entre um vértice $\in SA'$ e um $\notin SA'$. Finalmente, *Recursion* retorna os vértices candidatos a deslocamento incrementados em SA e o respectivo fluxo de caixa descontado DC.

Algoritmo 2: *Recursion* - versão original.

```
1  Recursion(newnode)
2      Initialize SA = {newnode}, DC = DC_newnode and CA = CA ∪ {newnode};
3      Do ∀i|i ∉ CA and i succeeds newnode in the current tree CT :
4          Recursion(i) → SA', DC'
5          if DC' ≥ 0 then
6              Set SA = SA ∪ SA' and DC = DC + DC';
7          else
8              CT = CT\(newnode, i);
9              Compute v_{k*l*} = min {f_l − d_l − f_k} and set CT = CT ∪ (k*, l*);
10                              (k*,l*)∈A
11                              k*∈SA'
12                              l*∉SA'
13             Do ∀ j ∈ SA' : set f_j = f_j + v_{k*l*};
14             Go to Step_3;
15     Do ∀i|i ∉ CA and i precedes newnode in the current tree CT :
16         Recursion(i) → SA', DC';
17         Set SA = SA ∪ SA' and DC = DC + DC';
18 Return
```

3.5.2 *Recursive Search*: um exemplo

Para exemplificar o comportamento de RS, pode-se utilizar o projeto representado na Figura 9(a) da seção 3.4, com os respectivos valores de duração (d_i) e fluxo de caixa (c_i). Com isso, assume-se que o projeto tem *deadline* (δ_n) no valor de 40 unidades de tempo e taxa de desconto de 2%. Assumindo também que ET está pronta (com datas antecipadas) e copiada em CT (*Current Tree*), as datas de início das atividades serão: $s_1 = 0,\ s_2 = 0,\ s_3 = 0,\ s_4 = 3,\ s_5 = 10,\ s_6 = 0,\ s_7 = 10,\ s_8 = 20$. Importa lembrar que para o algoritmo RS, o grafo com todas as restrições, Figura 9(a), não tem aresta entre os *dummies*, mas ET e CT têm, conforme Figura 10(a).

Com isso, a Figura 12 apresenta a sequência de cenas para o escalonamento

realizado por RS, através de seus componentes *Step_*3 e *Recursion*. Na cena (a), o algoritmo identifica a subárvore composta pelos vértices $\{2, 4\}$ na região cinza, remove a aresta (1,2), calcula a distância mais próxima fora da subárvore e insere uma aresta entre os vértices (4,7). A cena (a2) destaca o mesmo acontecimento, mas com ênfase ao esquema de empilhamento entre *Step_3* (representado pelo retângulo de fundo branco) e *Recursion* (representado pelo retângulo menor de fundo cinza). Em (a2), destaca-se a primeira chamada ao componente *Step_3* e, dentro dela, três chamadas ao componente *Recursion*. Ou seja, *Recursion* inicia o empilhamento a partir do vértice 1, que visita o 2, que visita o 4, que não tem sucessores nem predecessores não visitados. Desse modo, como a subárvore expressa por $SA = \{2, 4\}$ tem fluxo de caixa negativo, deve-se aplicar o deslocamento, com a remoção da aresta (1,2) (com um 'x' sobre ela) e a inserção da aresta (4,7).

Como houve deslocamento de uma subárvore, uma nova chamada ao componente *Step_3* ocorre. A segunda chamada está representada pelas cenas (b) e (b2) da Figura 12. Em (b), a subárvore $SA = \{2, 4\}$ é identificada com fluxo de caixa negativo. Desse modo, remove-se a aresta (1,3), com o indicativo de um 'x' sobre ela, calcula-se a distância mais próxima fora de SA' e se insere a aresta (5,8). Em (b2), é possível verificar que a pilha de *Recursion* inclui chamadas que visitam os vértices 1, 3 e 5 e que a subárvore expressa por SA é deslocada até o vértice 8, indicando ainda que a aresta $(k*, l*)$ é (5,8).

Como ocorreu o deslocamento de uma subárvore no empilhamento de *Recursion* na chamada corrente de *Step_3*, uma nova chamada ao mesmo componente *Step_3* é feita. Nesse caso, a terceira chamada de *Step_3* está representada pelas cenas (c) e (c2) na Figura 12. Em (c), destaca-se a subárvore $SA = \{2, 4\}$ identificada com fluxo de caixa negativo. Desse modo, remove-se a aresta (6,7), calcula-se a menor distância fora de SA' e a aresta (7,8) é inserida na CT. A cena (c2), por sua vez, mostra o esquema de empilhamento das chamadas de *Recursion* dentro da terceira execução de *Step_3*. Ou seja, os vértices 1, 6, 7, 4 e 2 são visitados nessa ordem e o algoritmo conclui que o conjunto $SA = \{7, 4, 2\}$ deve ser deslocado até o vértice 8.

Por fim, *Step_3* é executado pela quarta vez, já que houve deslocamento na chamada anterior. As cenas (d) e (d2) da Figura 12 destacam essa execução. No caso, a cena (d2) mostra que os vértices 1, 6, 8, 5, 3, 7, 4 e 2 são visitados nessa ordem no empilhamento de *Recursion*, que representa todo conjunto de vértices de CT. Importante notar também na cena (d) que, com os deslocamentos anteriores, todos os vértices com fluxo de caixa negativo passaram a ser vértices ascendentes ao *dummy* inicial. Assim, RS conclui o escalonamento com a seguinte agenda: $s_1 = 0, s_2 = 31, s_3 = 20, s_4 = 34, s_5 = 30, s_6 = 0, s_7 = 36, s_8 = 40$.

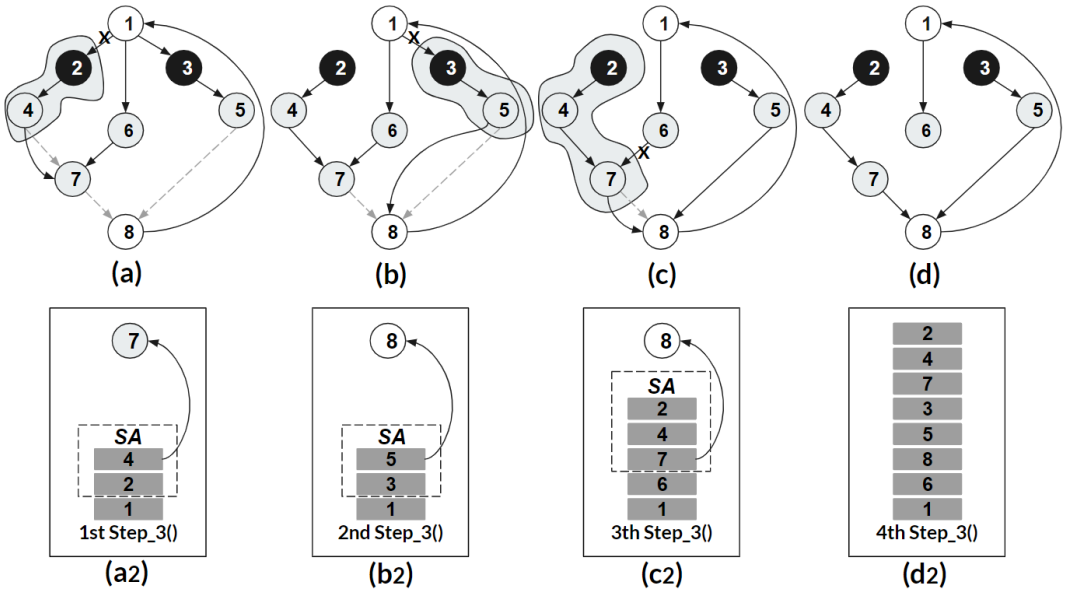

Figura 12 – RS exemplo.

3.5.3 *Steepest Ascent Approach*

Steepest Ascent Approach (SAA) é o segundo algoritmo fundamental nesta tese e foi proposto por Schwindt e Zimmermann (2001). Embora sua abordagem generalize a relação de precedência entre as atividades, uma adaptação é facilmente realizada, considerando apenas o tipo *finish-start* e *lag* zero, de acordo com o problema apresentado, como já foi feito em trabalho semelhante (VANHOUCKE; DEMEULEMEESTER; HERROELEN, 2000). Sua organização conta com três algoritmos componentes: *Steepest Ascent Direction* (SAD), *Vertex Ascent* (VA) e *Steepest Ascent Procedure* (SAP). Assim, o algoritmo componente SAD (de modo iterativo) busca por conjuntos de vértices (subárvores) com fluxo de caixa negativo na árvore geradora. O algoritmo componente VA identifica destinos para as subárvores encontradas com SAD, levando em consideração as restrições mais próximas de cada uma delas. O terceiro algoritmo componente SAP sintetiza a estratégia apresentada, incluindo chamadas para os dois primeiros algoritmos SAD e VA. Assim, SAP chama SAD e, quando subárvores são identificadas, VA é executado para realizar os deslocamentos. Desse modo, SAP itera enquanto subárvores forem identificadas com SAD. Sobre complexidade, os autores afirmam que SAD pode ser implementado em $O(n)$ e VA em $O(m\ log\ m)$ que equivale a $O(m\ log\ n^2) = O(2\ m\ log\ n) = O(m\ log\ n)$. No entanto, o desconhecimento do número de vezes em que se reinicia as buscas na árvore geradora é apontado como questão aberta, como ocorre em RS. **Isso também torna a complexidade global de SAA uma questão aberta**.

O pseudocódigo original de SAD segue indicado em Algoritmo 3. Entre os elementos desse algoritmo: a) z se refere a um vetor com valor binário para cada atividade i do conjunto

de vértices V; b) ϕ_i e c_i se referem à derivada parcial do fluxo de caixa descontado de cada atividade i; c) $C(i)$ se refere a um vetor utilizado para agrupar vértices sem sucessores ou predecessores no processo de busca por subárvores; d) V se refere a uma cópia dos vértices do projeto e; e) $Pred_G(i)$ e $Succ_G(j)$ são funções que verificam respectivamente predecessores e sucessores de um vértice.

Algoritmo 3: *Steepest Ascent Direction* (SAD) - versão original.

```
1  Set z_i := 0, φ_i := c_i, and C(i) := {i} for all i ∈ V.
2  while V ≠ {0}
3      if V contains a node i ≠ 0 with Pred_G(i) ∩ V = ∅ possessing at most one sucessor
         j ∈ V then
4          V := V \ {i}.
5          if Succ_G(i) ∩ V = {j} then
6              φ_j := φ_j + φ_i, C(j) := C(j) ∪ C(i).
7      else
8          Determine node j ∈ V, j ≠ 0 with Succ_G(j) ∩ V = ∅ possessing exactly one predecessor
             i ∈ V.
9          V := V \ {j}.
10         if φ_j > 0 then z_h := 1 for all h ∈ C(j).
11         else φ_i := φ_i + φ_j, C(i) ← C(i) + C(j).
```

O pseudocódigo de VA está indicado em Algoritmo 4. Entre os seus elementos, estão: a) U que se refere aos vértices que devem ser deslocados. Para isso, o algoritmo verifica no vetor binário z quais vértices de V têm valor igual a 1 (indicação de que o vértice deve ser deslocado); b) E_G que se refere ao conjunto de arestas da árvore geradora; c) as linhas 4 e 5 que se referem a determinação da aresta (k, l) que expressa a menor distância entre um vértice $k \in U$ e um vértice $l \notin U$. Assim, S se refere ao início das atividades e δ equivale à duração de k, quando o problema trata apenas da relação *finish-start* com $lag = 0$, com exceção de $l = dummy$ final e $k = dummy$ inicial. No caso da exceção, $\delta = -deadline$ do projeto e; d) G que se refere à árvore geradora, neste algoritmo.

Algoritmo 4: *Vertex Ascent* (VA) - versão original.

```
1  U := {j ∈ V | z_j = 1}.
2  for (i, j) ∈ E_G with j ∉ C(i) and i ∉ C(j) do E_G := E_G \ {(i, j)}.
3  while U ≠ ∅
4      Determine (k, l) ∈ E with k ∈ U and l ∈ V\U such that
5      S_l − S_k − δ_kl = σ := min_{k'∈U} min_{(k', l') ∈ E, l'∈V\U} (S_l' − S_k' − δ_k'l').
6      Determine node set C(j) of the weak component of G with k ∈ C(j).
7      for h ∈ C(j) do S_h + σ.
8      U := U \C(j).
9      E_G := E_G ∪ {(k, l)}.
```

Por fim, o pseudocódigo de SAP está indicado em Algoritmo 5. Desse modo, a) ES (*Early Schedule*) se refere a um vetor com as datas antecipadas quanto possível das atividades na árvore geradora; b) S, que sofre alterações dos deslocamentos aplicados, se

refere a uma cópia de ES; c) e z se refere a um vetor que tem sua quantidade testada de itens com valor igual a 0. Nesse ponto, deve-se destacar que como z é um vetor e não um escalar, o teste da linha 6 do pseudocódigo original de SAP (Algoritmo 5) não parece ser a forma mais legível de expressar um teste condicional de quantidade de elementos com um determinado valor de interesse, no caso, igual a 0. Por isso, a versão implementada nesta tese para esse algoritmo (que será apresentada adiante) utiliza uma outra estratégia.

Algoritmo 5: *Steepest Ascent Procedure* (SAP) - versão original.

```
1  Determine schedule ES and a corresponding initial spanning tree G = (V, E_G).
2  Set S := ES.
3  Repeat
4     Determine optimal direction z by applying Algorithm SAD to schedule S and
5     spanning tree G.
6     if z = 0 then return S.
7     else
8        Ascent to new vertex S' by applying Algorithm VA to schedule S, spanning
9        tree G, and steepest ascent direction z.
10       Set S := S'.
```

3.5.4 *Steepest Ascent Approach*: um exemplo

Para exemplificar o comportamento de *Steepest Ascent Approach* (SAA), foi considerado o mesmo projeto utilizado para RS, conforme Figura 9 da seção 3.4. Assim, o primeiro componente de SAA, ou seja, *Steepest Ascent Procedure* (SAP) faz a antecipação das atividades da árvore geradora. Com as antecipações, as atividades têm a seguinte agenda de início: $S_1 = 0,\ S_2 = 0,\ S_3 = 0,\ S_4 = 3,\ S_5 = 10,\ S_6 = 0,\ S_7 = 10,\ S_8 = 20$. Vale lembrar que (diferente de RS) SAA utiliza uma árvore geradora sem aresta entre os *dummies*, conforme Figura 10(b), mas acrescenta tal aresta no grafo com todas as precedências do projeto, conforme Figura 9(b).

Na cena (b), os vértices 4, 7 e 8 são identificados como vértices tipo *sumidouro* (*sink*) com predecessores únicos e fluxo de caixa descontado positivo. Com isso, tais vértices são agrupados em V, através do conjunto $C(i)$, em seus respectivos predecessores, incluindo seus fluxos de caixa. Isso faz de $C(2) = \{2, 4\}, C(6) = \{6, 7\}$ e $C(8) = \{8, 5\}$. Depois disso, os vértices 4, 7 e 8 são eliminados de V.

Na cena (c), o número de vértices de V diminui e o algoritmo encontra os vértices 2, 6 e 5 como tipo *sumidouro* (*sink*) com predecessores únicos. Os vértices 5 e 6 têm fluxo de caixa descontado positivo, levando em consideração o valor acumulado dos vértices agrupados na cena anterior e, por isso, são agrupados em seus predecessores e depois removidos de V. Mas o vértice 2, considerando o valor acumulado da cena anterior, tem fluxo de caixa descontado negativo e, por isso, é memorizado no conjunto U como candidato

a deslocamento, fazendo de $U = \{\{2, 4\}\}$. Assim, os vértices 2, 6 e 5 são eliminados de V.

Na cena (d), o vértice 3 é identificado como *sumidouro*, agregado em seu predecessor, memorizado como candidato a deslocamento, pois o fluxo de caixa descontado agregado é negativo, depois removido de V. Isso faz de $U = \{\{2, 4\}, \{3, 5, 8\}\}$. Na cena (e), V contém apenas o vértice 1, fazendo com que o laço principal de SAD pare de iterar e U passe a conter as duas subárvores expressas por $\{2, 4\}$ e $\{3, 5, 8\}$.

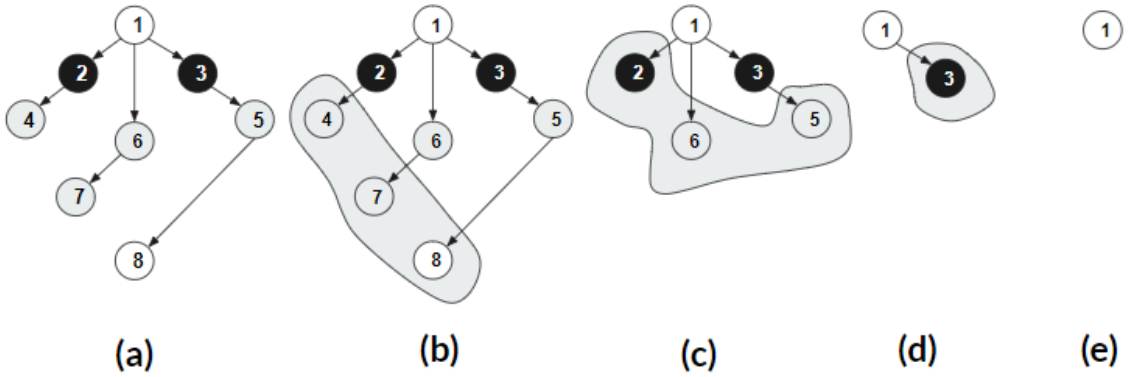

Figura 13 – Primeira execução de SAD.

Como SAD retornou uma direção apontando subárvores que devem ser deslocadas, o componente VA é executado para efetivar os deslocamentos. Desse modo, a primeira execução de VA pode ser apresentada pela cena (a) da Figura 14. Ainda na cena (a), as subárvores que fazem parte de U estão destacadas nas duas zonas cinzas. Com isso, VA faz duas remoções de arestas, representadas pelo 'x' sobre (1,2) e (1,3). Ainda na cena (a), procura-se entre os vértices das subárvores $\in U$ qual deles tem a menor distância entre cada data de fim $(S_k - d_k)$ e a data de início (S_l) de um sucessor $\notin U$. Nesse caso, os vértices $\in U$ se referem às subárvores dos conjuntos $C(2) = \{2, 4\}$ e $C(3) = \{3, 5, 8\}$, destacadas nas zonas cinzas. Assim, a menor distância entre os vértices de $C(2)$ e um vértice $\notin U$ está entre os vértices 4 e 7. Ou seja, a menor distância $\notin U$ que $C(2)$ tem é $S_7 - S_4 + d_4 = 10 - 3 + 5 = 5$, que passa a ser seu passo de deslocamento (σ). No caso da subárvore $C(3)$, a menor distância $\notin U$ está entre os vértices 8 e 1, que contém um *lag* negativo do tamanho do *deadline*, expresso pela aresta cinza tracejada. Então, a menor distância de $C(3)$ é dada por $S_1 - S_8 - (-\delta_n) = 0 - 20 - (-40) = 20$, que passa a ser seu passo de deslocamento (σ). Como VA desloca por ordem de menor distância, a primeira delas é $C(2)$, que tem $\sigma = 5$, levando em consideração as datas correntes de início das atividades expressas em $S = (0, 0, 0, 3, 10, 0, 10, 20)$. Por isso, as atividades de $C(2)$ são incrementadas com tal passo: $S_4 = S_4 + 5 = 8$; $S_2 = S_2 + 5 = 5$. Depois disso, $C(2)$ é retirada de U. A última subárvore $\in U$ passa a ser $C(3)$, que tem o passo de deslocamento $\sigma = 20$. Então, suas atividades são incrementadas: $S_3 = S_3 + 20 = 20$; $S_5 = S_5 + 20 = 30$; $S_8 = S_8 + 20 = 40$. Depois disso, $C(3)$ é removida de U, que se torna $\varnothing$. Desse modo, VA retorna uma solução

com o seguinte agendamento $S = (0, 5, 20, 8, 30, 0, 10, 40)$.

A Figura 14(b) destaca a condição resultante da árvore geradora após as remoções e inserções de arestas da primeira execução de VA. A árvore da Figura 14(c) é isomórfica à árvore da Figura 14(b). Sua utilidade está no destaque dado às posições dos vértices 8, 5 e 3, que com o deslocamento passaram a ser ascendentes ao vértice 1 (o *dummy* inicial).

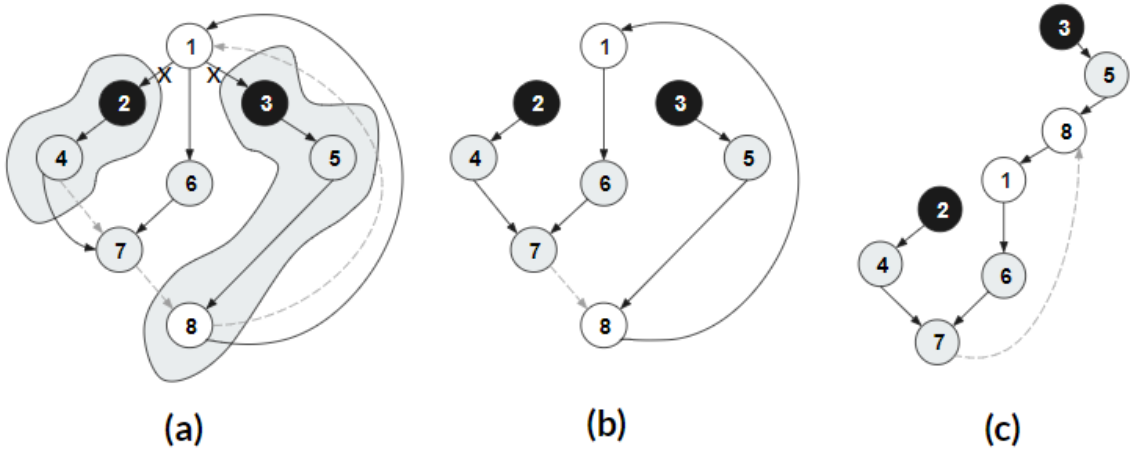

Figura 14 – Primeira execução de VA (SAA) e *Shift_activities* (HS).

Como a primeira execução de SAD identificou uma direção que melhorou a função objetivo e VA deslocou, a segunda execução de SAD ocorre em busca de uma nova direção. Nesse caso, a Figura 15 indica a sequência de cenas da segunda execução de SAD. Na cena (a), tem-se a árvore geradora que destaca a posição ascendente do *dummy* inicial.

Na cena (b), os vértices 2 e 3 são identificados como $fonte$ (*source*) de sucessores únicos e, por causa disso, são agregados respectivamente em $C(4) = \{4, 2\}$ e $C(5) = \{5, 3\}$ com seus fluxos de caixa. Depois disso, os vértices 2 e 3 são removidos de V e o conjunto U segue $\varnothing$, independente dos valores de fluxo de caixa, pois apenas vértices tipo *sumidouro* podem ser memorizados em U.

Na cena (c), os vértices 4 e 5 são agregados em seus respectivos sucessores, pela mesma razão da cena (b), tornando $C(7) = \{7, 4, 2\}$, $C(8) = \{8, 5, 3\}$ e U segue $\varnothing$.

Na cena (d), o vértice 8 é um $fonte$ (*source*) que tem $C(8) = \{8, 5, 3\}$ e deve ser agregado em seu único sucessor, tornando $C(1) = \{1, 8, 5, 3\}$. Já o vértice 7 é um *sumidouro* que tem $C(7) = \{7, 4, 2\}$ com fluxo de caixa agregado descontado negativo. Isso faz com que $C(7)$ seja memorizada em U, tornando $U = \{\{7, 4, 2\}\}$. Assim, os vértices 7 e 8 são removidos de V.

Na cena (e), o vértice 6 é identificado como *sumidouro* de predecessor único, com $C(6) = \{6\}$ e com fluxo de caixa descontado positivo, o que o faz ser removido de V. Na cena (f), V tem apenas o vértice 1, fazendo com que SAD retorne o conjunto U com os vértices da subárvore $C(7) = \{7, 4, 2\}$.

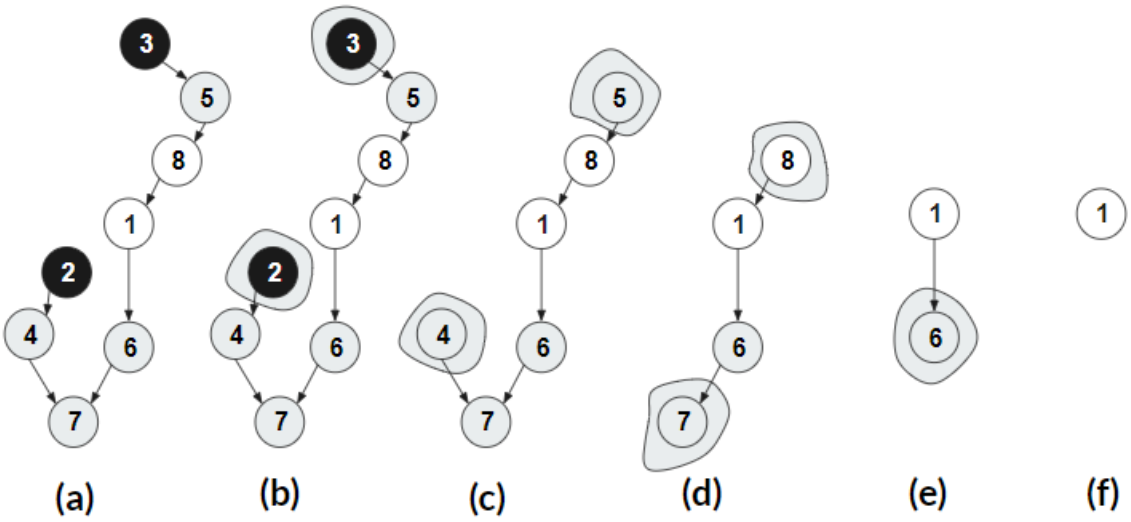

Figura 15 – Segunda execução de SAD.

A segunda execução de VA está destacada na Figura 16. A cena (a) da Figura 16 mostra que a aresta (6,7) é removida, assim como a menor e única distância entre um vértice $\in U$ e um vértice $\notin U$ está entre os vértices 7 e 8, indicado pela aresta tracejada na cor cinza. Nesse caso, lembrando que a solução corrente dentro da região viável tem os valores de início das atividades de árvore geradora expressos em $S = (0, 5, 20, 8, 30, 0, 10, 40)$, o passo de deslocamento é dado por $\sigma = S_8 - S_7 + d_7 = 40 - 10 + 4 = 26$. Isso faz com que os vértices da única subárvore em U, $C(7) = \{7, 4, 2\}$, tenham suas datas de início incrementadas com tal passo de deslocamento ($\sigma = 26$). Ou seja, $S_7 = S_7 + 26 = 36; S_4 = S_4 + 26 = 34; S_2 = S_2 + 26 = 31$. Além disso, inclui-se a aresta (7,8), indicada na cena (a).

Por fim, a cena (b) da Figura 16 destaca que a subárvore $C(7) = \{7, 4, 2\}$ passou a ter posição de ascendência em relação ao *dummy* inicial. Então, VA termina e retorna uma nova solução expressa em $S = (0, 31, 20, 34, 30, 0, 36, 40)$.

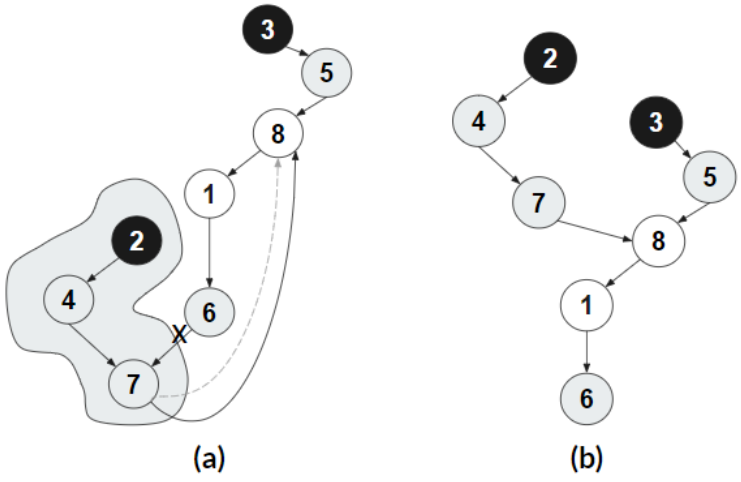

Figura 16 – Segunda execução de VA (SAA) e *Shift_activities* (HS).

Como SAD encontrou uma direção na segunda execução e VA deslocou, uma terceira execução de SAD é realizada. A Figura 17 apresenta a sequência de cenas dessa terceira execução. A cena (a) apresenta a estrutura inicial de V.

A cena (b) destaca que os vértices 2 e 3 são identificados como $fonte$ e, por isso, são agrupados em seus respectivos sucessores exclusivos e removidos de V, enquanto o conjunto U segue $\varnothing$. Na mesma cena (b), o vértice 6, com $C(6) = \{6\}$, é identificado como *sumidouro* e é agrupado em seu predecessor exclusivo, fazendo de $C(1) = \{1, 6\}$, além de ser removido de V. No entanto, $C(1) = \{1, 6\}$ tem fluxo de caixa descontado positivo e, por isso, U segue $\varnothing$.

Na cena (c), os vértices 4 e 5 são identificados como $fonte$ e são agrupados em seus respectivos sucessores únicos, fazendo $C(7) = \{7, 4, 2\}$ e $C(8) = \{8, 5, 3\}$, além de serem removidos de V. Como 4 e 5 são vértices tipo $fonte$, independente do fluxo de caixa, o conjunto U segue $\varnothing$.

Na cena (d), o vértice 7 é identificado como $fonte$ e agrupado no vértice 8, fazendo $C(8) = \{8, 5, 3, 7, 4, 2\}$, além de ser removido de V. Mais uma vez U segue $\varnothing$.

Na cena (e), o vértice 8 é identificado como $fonte$ e agrupado no vértice 1, fazendo de $C(1) = \{1, 2, 3, 4, 5, 6, 7, 8\}$, além de ser removido de V. Nesse caso, U continua $\varnothing$.

Na cena (f), V contém apenas o *dummy* inicial e SAD conclui suas verificações, com o retorno de $U = \varnothing$. Então, como não foi encontrada uma nova direção para melhorar o função objetivo, SAP conclui seu trabalho e retorna $S = (0, 31, 20, 34, 30, 0, 36, 40)$ como solução definitiva do escalonamento.

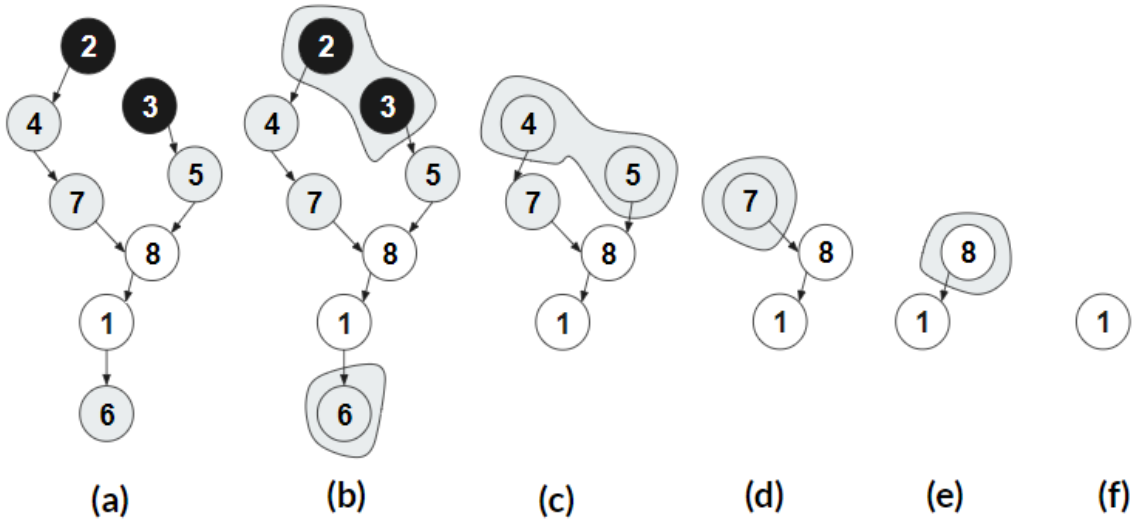

Figura 17 – Terceira execução de SAD.

3.5.5 *Hybrid Search*

Hybrid Search (HS) é o terceiro algoritmo fundamental deste trabalho e foi proposto por Vanhoucke (2006). Sua abordagem é uma combinação das estratégias dos algoritmos RS e SAA. Sua organização conta com três algoritmos componentes: *Recursion*, *Shift_activities* e *Hybrid Recursive Search* (HRS). O primeiro deles, chamado *Recursion*, identifica subárvores que podem ser deslocadas, através de uma busca em profundidade recursiva. Já o algoritmo componente *Shift_activities* encontra destinos para as subárvores e realiza os deslocamentos, respeitando as restrições de precedência. O algoritmo componente HRS sintetiza toda abordagem de escalonamento, invocando os componentes *Recursion* e *Shift_activities*. Desse modo, a execução de HRS dura enquanto subárvores são identificadas.

Embora a estratégia de busca também seja recursiva como a de RS, o algoritmo HS tem a capacidade de identificar vários conjuntos de vértices antes de realizar deslocamentos, assim como o algoritmo SAA (de modo iterativo). Portanto, HS quando parte para a realização de deslocamento, vários conjuntos de vértices podem ter sido identificados na última busca realizada, assim como em SAA.

Além disso, HS inclui a estratégia de inversão de busca e deslocamento, quando existe mais da metade dos vértices com fluxo de caixa negativo no projeto. Quando esse é o caso, HS cria uma *late tree* atrasando as atividades o máximo possível (respeitando as restrições de precedência) e tomando o *deadline* como referência para o começo das buscas ao invés do *dummy* inicial. Nesse caso, a busca recursiva sempre é iniciada no *dummy* final (com data de início igual ao *deadline*) e os conjuntos de vértices identificados com fluxo de caixa positivo são deslocados na direção do *dummy* inicial. **Sobre complexidade, não existem considerações para HS, o que torna esse aspecto uma questão aberta**.

O pseudocódigo de *Recursion* de HS está indicado em Algoritmo 6. No caso, *Recursion* recebe como argumento um vértice (*newnode*) e assume que CA, ST e SS são estruturas globais. Desse modo, CA se refere às atividades visitadas (*Considered Activities*), ST se refere à árvore geradora (de *Spanning Tree*), SS se refere ao conjunto de subárvores identificadas (*Set of SA*) e DC se refere ao fluxo de caixa descontado (*Discounted Cash Flow*). Por fim, *Recursion* retorna SA e DC.

Algoritmo 6: *Recursion* de HS - versão original.

Sub-procedure *Recursion*($newnode$)
  $SA = \{newnode\}, DC = DC_{newnode}$ and $CA = CA \cup \{newnode\}$
  Do $\forall i | i \notin CA$ and i succeeds $newnode$ in the tree ST :
    $Recursion(i) \rightarrow SA', DC'$
    if $DC' \geq 0$ **then** set $SA = \text{SA} \cup SA'$ and $DC = DC + DC'$
    else $ST = ST \backslash (newnode, i)$ and $SS = SS \cup SA'$
  Do $\forall i | i \notin CA$ and i precedes $newnode$ in the tree ST :
    $Recursion(i) \rightarrow SA', DC'$
    Set $SA = \text{SA} \cup SA'$ and $DC = DC + DC'$
Return

O pseudocódigo de *Shift_activities* está indicado em Algoritmo 7. Esse algoritmo recebe como argumento SS, referente às subárvores que devem ser deslocadas. No caso, os vértices das subárvores de SS são desmembrados e armazenados em Z. *Shift_activities* também inclui a função *Compute* v_{k*l*} que identifica a menor distância entre os vértices $\in Z$ e os $\notin Z$. Desse modo, v_{k*l*} expressa o passo de deslocamento incrementado no início de cada atividade que deve ser movida.

Algoritmo 7: *Shift_activities* - vesão original.

Sub-procedure *Shift_activities*(SS)
  Let $Z = \{i \in SA | SA \in SS\}$ be the set of activities which can possibly be delayed;
  while $Z \neq \varnothing$ **do**
    Compute $v_{k*l*} = \min_{\substack{(k*,l*) \in A \\ k* \in Z \\ l* \notin Z}} \{s_l - s_k - l_{kl}\}$
    Do $\forall\ i \in SA | k* \in SA$: Set $s_i = s_i + v_{k*l*}$ and $Z = Z \backslash \{i\}$;
    Set $ST = ST + (k*, l*)$;
Return

Por fim, o pseudocódigo do componente *Hybrid Recursive Search* (HRS) está indicado em Algoritmo 8. No caso, HRS inicia uma busca em profundidade através de *Recursion* e continua sua execução enquanto subárvores (SS) forem identificadas e consequentemente deslocadas. Quando não for o caso, o fluxo de caixa descontado global do projeto é reportado com o valor presente líquido máximo.

Algoritmo 8: *Hybrid Recursive Search* (HRS) - versão original.

```
1 Procedure Hybrid_recursive_search
2     CA = SS = ∅
3     Do Recursion(1)
4     if SS ≠ ∅ then
5         Shift_activities(SS) and reapet Recursion(1)
6     else
7         Report the optimal solution DC'
8 Return
```

3.5.6 *Hybrid Search*: um exemplo

O mesmo projeto para exemplificar os comportamentos de RS e SAA, indicado na Figura 9, também foi utilizado para HS. No caso de HS, as duas primeiras ações do algoritmo são: a) tornar vazias as memórias CA e SS e; b) realizar a busca em profundidade com *Recursion*. Assim como em SAA, em HS se utiliza o grafo da Figura 9 para verificação de restrições. No caso das antecipações, o algoritmo gera o mesmo agendamento de início de RS e SAA, ou seja, as atividades de ST têm as seguintes datas: $s_1 = 0,\ s_2 = 0,\ s_3 = 0,\ s_4 = 3,\ s_5 = 10,\ s_6 = 0,\ s_7 = 10,\ s_8 = 20$. Sobre a árvore geradora, HS utiliza a estrutura indicada pela Figura 13(b).

A primeira busca em profundidade é iniciada a partir da execução de HRS com a chamada ao algoritmo componente *Recursion*. Tal busca em profundidade está caracterizada nas duas cenas da Figura 18. A cena (a) indica que as subárvores $SA_1 = \{2, 4\}$ e $SA_2 = \{3, 5, 8\}$ foram identificadas como candidatas a deslocamento e são memorizadas em SS. Já a cena (b) indica a mesma busca, mas com destaque para o empilhamento das chamadas de *Recursion* e as duas subárvores identificadas SA_1 e SA_2.

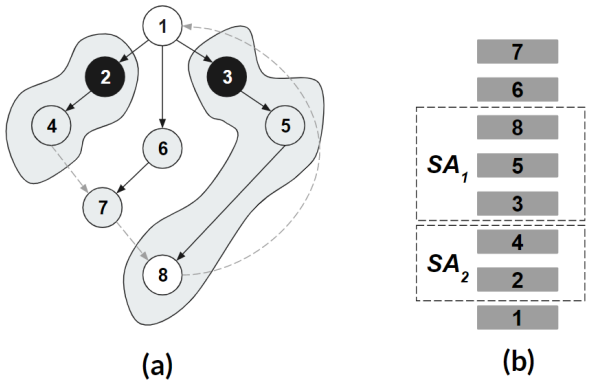

Figura 18 – Primeira busca em profundidade em HS.

Como a primeira busca em profundidade encontrou duas subárvores para deslocamento, o algoritmo componente *Shift_activities* é executado para efetivar os deslocamentos.

Desse modo, assim como ocorre no algoritmo componente VA de SAA, em *Shift_actitivities*, as arestas (1,2) e (1,3) são removidas. Além disso, as arestas (4,7) e (8,1) são inseridas, indicando as respectivas restrições mais próximas das subárvores identificadas. Os vértices das subárvores $SA_1 = \{2, 4\}$ e $SA_2 = \{3, 5, 8\}$ são incrementados com os respectivos passos de deslocamento $v_{k*l*} = 5$ e $v_{k*l*} = 20$. Desse modo, assim como em SAA, a Figura 14(b) indica a condição resultante da árvore geradora, após a primeira execução de *Shift_activities*. Vale relembrar que a árvore da Figura 14(c) é isomórfica à árvore da Figura 14(b).

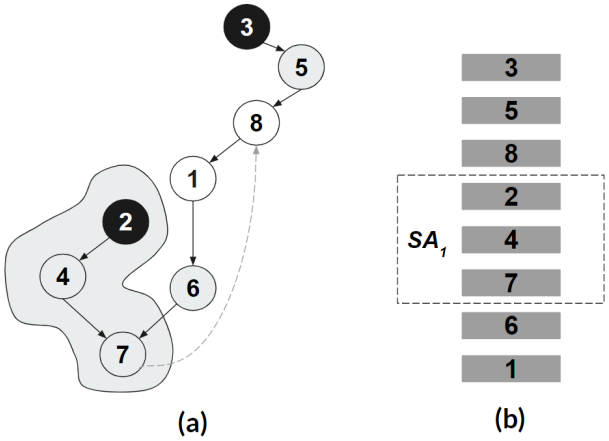

Figura 19 – Segunda busca em profundidade em HS.

Como a primeira busca em profundidade identificou subárvore e *Shift_activities* deslocou, uma segunda busca em profundidade é feita. Nesse caso, a Figura 19(a) mostra o resultado da segunda busca que identifica a subárvore com os vértices $\{2, 4, 7\}$ como um novo conjunto candidato a deslocamento. A cena (b) da Figura 19 destaca o empilhamento das chamadas da segunda busca com *Recursion* e a subárvore identificada.

Com isso, *Shift_activities* é executado uma segunda vez para efetivar os deslocamentos. Assim como VA de SAA fez, *Shift_actitivities* também remove a aresta (6,7), como destaca o 'x' na cena (a) da Figura 16. Então, a aresta inserida (7,8) indica a restrição mais próxima da subárvore identificada. Os vértices da subárvore $\{2, 4, 7\}$ são incrementados com o passo de deslocamento $v_{k*l*} = 26$. Desse modo, a Figura 16(b) mostra a condição resultante da árvore geradora.

Como a segunda busca também identificou subárvore e *Shift_activities* efetivou os deslocamentos, uma terceira busca é feita. Nesse caso, a Figura 20(a) mostra o resultado da terceira busca, que não encontra qualquer subárvore candidata a deslocamento. A cena (b) da Figura 20 mostra a ordem do empilhamento das chamadas de *Recursion* da terceira busca. Então, como nenhuma subárvore é identificada, HS conclui o trabalho de escalonamento com a seguinte agenda para as atividades na árvore geradora: $s_1 = 0$, $s_2 = 31$, $s_3 = 20$, $s_4 = 34$, $s_5 = 30$, $s_6 = 0$, $s_7 = 36$, $s_8 = 40$.

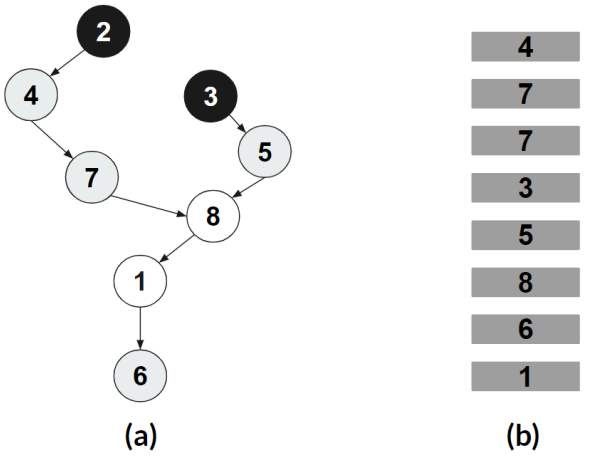

Figura 20 – Terceira busca em profundidade em HS.

4 ALGORITMOS DO EXPERIMENTO

Este capítulo apresenta as versões implementadas dos algoritmos utilizados no experimento desta tese. Isso inclui a distinção entre versões $forward$ e *backward* de cada um dos componentes implementados. Inclui também o pseudocódigo implementado que corresponde à função de cálculo de menor distância contida em todos os algoritmos originais como caixa preta. Além disso, são destacadas algumas diferenças de conveniência ou estilo entre as versões implementadas e as originais. É válido justificar também que os pseudocódigos das versões implementadas estão escritos em inglês para que apenas as diferenças em relação às versões implementadas estejam em evidência.

4.1 *RECURSIVE SEARCH FORWARD-BACKWARD*

O primeiro algoritmo implementado para o experimento foi *Recursive Search Forward-Backward* (RSFB), variação do algoritmo *Recursive Search* (RS), que incorpora a estratégia de inversão de busca e deslocamento quando o projeto tem mais da metade das atividades com fluxo de caixa negativo. Essa variação é inspirada na mesma estratégia contida no algoritmo *Hybrid Search* (HS) de Vanhoucke (2006). Isso significa que RSFB escolhe a direção da busca e deslocamento, considerando o percentual de atividades com fluxo de caixa negativo: a) se o projeto tem até 50% de atividades com fluxo de caixa positivo, a busca e o deslocamento são do tipo *forward* (do início para o fim); b) caso contrário, a busca e o deslocamento são do tipo *backward* (do fim para o início).

Então, quando RSFB escolhe a direção *backward*, o algoritmo utiliza uma *late tree* ao invés de uma *early tree* como árvore geradora inicial. Assim, a busca em profundidade é iniciada a partir do último vértice (*dummy* final) e as subárvores candidatas a deslocamento têm fluxo de caixa positivo ao invés de negativo. Quando identificadas, as subárvores são deslocadas na direção do primeiro vértice (*dummy* inicial) ao invés do último. Essa lógica é um espelho da lógica tipo *forward* apresentada no capítulo anterior.

4.1.1 Diferenças em *Step_3*

No componente *Step_3* de RSFB, a diferença entre $forward$ e *backward* está destacada na linha 3 dos respectivos pseudocódigos das versões implementadas em Algoritmos 9 e 10. A referência de início para $forward$ é o *dummy* inicial (expresso por 1) e a referência de início para *backward* é o *dummy* final (expresso por n).

Sobre diferenças de conveniência ou estilo das versões implementadas (Algoritmos 9 e 10) para a versão original de *Step_3* (Algoritmo 1), deve ser destacado que as versões implementadas incluem a variável denominada *total_Step_3* (linha 2) que tem propósito de contar o número de reinícios de busca na árvore geradora. Nesse caso, assume-se que *total_Step_3* foi iniciada uma única vez com valor zero imediatamente antes da primeira chamada ao componente *Step_3*. Essa contagem se refere a uma das métricas discutidas no Capítulo 6 que trata do experimento. Além disso, as versões implementadas de *Step_3* destacam CA como uma estrutura de dados global.

Algoritmo 9: *Step_3 - Forward.*

```
1 procedure Step_3() {CA is a global structure}
2     total_Step_3 = total_Step_3 + 1
3     CA ← ∅
4     SA', DC' ← Recursion(1)
5     Report the optimal solution DC'
```

Algoritmo 10: *Step_3 - Backward.*

```
1 procedure Step_3() {CA is a global structure}
2     total_Step_3 = total_Step_3 + 1
3     CA ← ∅
4     SA', DC' ← Recursion(n)
5     Report the optimal solution DC'
```

4.1.2 Diferenças em *Recursion*

No caso do componente *Recursion* de RSFB, as diferenças entre $forward$ e *backward* estão apontadas nas linhas 4, 9, 10, 11, 12, 13, 14 e 18 dos pseudocódigos em Algoritmos 11 e 12. É possível notar que tais linhas expressam referências de busca e deslocamento opostas (entre as versões $forward$ e *backward*).

Sobre diferenças de conveniência ou estilo das versões implementadas (Algoritmos 11 e 12) para a versão original de *Recursion* (Algoritmo 2), destaque para:

- Nas versões implementadas, o retorno é explícito (linha 22);
- Nas versões implementadas, CA e CT são indicadas como estruturas globais (linha 1);
- Nas versões implementadas, G se refere ao grafo que inclui todas restrições de precedência do projeto (linha 11);
- Nas versões implementadas, foi incluída a variável *total_Recursion* (linha 2) utilizada para contar o número de chamadas recursivas. Essa é uma variável global que é iniciada com valor zero uma única vez imediatamente antes da primeira chamada ao

componente *Step_3*. Essa contagem também se refere a uma das métricas discutidas no Capítulo 6 que trata do experimento.

Algoritmo 11: *Recursion - Forward.*

function *Recursion*(*newnode*) {*CA, CT are global structures*}
  total_Recursion = *total_Recursion* + 1
  $SA \leftarrow \{newnode\}; DC \leftarrow DC_{newnode}; CA \leftarrow CA + newnode$
  for $(i | i \notin CA$ **and** i succeeds $newnode \in CT)$ **do**
    $SA', DC' \leftarrow Recursion(\text{i})$
    if $DC' \geq 0$ **then**
      $SA \leftarrow SA + SA'; DC \leftarrow DC + DC'$
    else
      $CT \leftarrow CT - (newnode, i)$
      Compute $v_{k*l*} = \min_{\substack{(k*,l*)\in G \\ k*\in SA \\ l*\notin SA}} \left\{ f_l - d_k - f_k \right\}; CT \leftarrow CT + (k*, l*)$
      $\forall\, j \in SA' : f_j \leftarrow f_j + v_{k*l*}$
      Step_3()
    end
  end
  for $(i | i \notin CA$ **and** i precedes $newnode \in CT)$ **do**
    $SA', DC' \leftarrow Recursion(i)$
    $SA \leftarrow SA + SA'; DC \leftarrow DC + DC'$
  end
  return (SA, DC)

Algoritmo 12: *Recursion - Backward.*

function *Recursion*(*newnode*) {*CA, CT are global structures*}
  total_Recursion = *total_Recursion* + 1
  $SA \leftarrow \{newnode\}; DC \leftarrow DC_{newnode}; CA \leftarrow CA + newnode$
  for $(i | i \notin CA$ **and** i precedes $newnode \in CT)$ **do**
    $SA', DC' \leftarrow Recursion(\text{i})$
    if $DC' \geq 0$ **then**
      $SA \leftarrow SA + SA'; DC \leftarrow DC + DC'$
    else
      $CT \leftarrow CT - (i, newnode)$
      Compute $v_{l*k*} = \min_{\substack{(l*,k*)\in G \\ k*\notin SA \\ l*\in SA}} \left\{ s_k - f_l \right\}; CT \leftarrow CT + (l*, k*)$
      $\forall\, j \in SA' : f_j \leftarrow f_j - v_{l*k*}$
      Step_3()
    end
  end
  for $(i | i \notin CA$ **and** i succeeds $newnode \in CT)$ **do**
    $SA', DC' \leftarrow Recursion(i)$
    $SA \leftarrow SA + SA'; DC \leftarrow DC + DC'$
  end
  return (SA, DC)

4.2 *STEEPEST ASCENT APPROACH FORWARD-BACKWARD*

O segundo algoritmo implementado para o experimento, *Steepest Ascent Approach Foward-Backward* (SAAFB), refere-se a uma variação do algoritmo *Steepest Ascent Approach* (SAA), com a incorporação da estratégia de inversão de busca e deslocamento, também inspirada no algoritmo *Hybrid Search* (HS). Assim, as referências de $forward$ e *backward* seguem destacadas em pseudocódigos distintos para cada componente de SAAFB.

4.2.1 Diferenças em *Steepest Ascent Direction* (SAD)

No caso do componente *Steepest Ascent Direction* (SAD) de SAAFB, as diferenças entre $forward$ e *backward* estão apontadas nas linhas sublinhadas (5, 9 e 11) nos pseudocódigos em Algoritmos 13 e 14. Essas linhas expressam referências opostas em relação às buscas que SAD pode fazer (entre as versões $forward$ e *backward*).

Sobre as principais diferenças de conveniência ou estilo das versões implementadas (Algoritmos 13 e 14) para a versão original de SAD (Algoritmo 3), destacam-se:

- Nas versões implementadas, ST e seus vértices (V_{st}) são indicados como estruturas globais (linha 1);
- Nas versões implementadas, V é indicado explicitamente como cópia de V_{st}, especialmente por V sofrer remoções de vértices nas linhas 7 e 12;
- Nas versões implementadas, as condicionais das linhas 5 e 9 (Algoritmos 13 e 14) foram simplificadas em relação à versão do algoritmo original. No caso da primeira condicional na versão original (linha 3 em Algoritmo 3), o algoritmo testa se há algum vértice $i \neq$ do *dummy* inicial $\in V$, tal que o conjunto de vértices predecessores de $i \cap V = \varnothing$. Mas como consta na linha 5 de cada versão implementada (Algoritmos 13 e 14) é suficiente testar a existência de vértice tipo $fonte$ (*source*) $\in V \neq$ do *dummy* inicial com somente um sucessor. No caso da segunda condicional na versão original (linha 8 em Algoritmo 3), testa-se a existência de algum vértice denominado $j \neq$ do *dummy* inicial $\in V$, tal que o conjunto de vértices predecessores $\cap V = \varnothing$. Do mesmo modo, como consta nas versões implementadas (linha 9 em Algoritmos 13 e 14), é suficiente testar se existe vértice tipo *sumidouro* (*sink*) denominado $j \in V \neq$ do *dummy* inicial com apenas um predecessor;
- As versões implementadas não incluem a condicional da linha 5 contida na versão original (Algoritmo 3), uma vez que o único sucessor de i é j;

- Nas versões implementadas, ao invés de usar um vetor binário para memorizar os vértices que devem ser deslocados (conforme linha 10 da versão original em Algoritmo 3), é suficiente memorizar todo o conjunto de vértices a ser deslocado em $C(j)$ (linha 11 em Algoritmos 13 e 14);

- Nas versões implementadas, há retorno explícito (linha 16);

- Nas versões implementadas, a derivada parcial do fluxo de caixa (linha 3 em Algoritmos 13 e 14) é apresentada de modo mais detalhado do que na versão original (linha 1 em Algoritmo 3). Desse modo, a linha 3 das versões implementadas incluem c_i como fluxo de caixa não descontado, S_i como o início da atividade i e d_i como a duração. Então, a derivada parcial (representada por ϕ_i na versão original) de uma atividade i pode ser dada pela expressão abaixo. Onde $c_i\ e^{-\alpha^{(S_i+d_i)}}$ representa o fluxo de caixa descontado, $e^{-\alpha^{(S_i+d_i)}} = \frac{1}{(1+r)^{S_i+d_i}}$ representa o fator de desconto e r a taxa de desconto;

$$\frac{d(c_i\ e^{-\alpha(S_i+d_i)})}{d(S_i+d_i)} = -\alpha\ c_i\ e^{-\alpha^{(S_i+d_i)}} \tag{4.1}$$

- Nas versões implementadas, foi inserida a variável *iteration_SAD* (iniciada com valor zero no mesmo instante que o algoritmo SAA também é iniciado) que é utilizada para contar todas as iterações que identifiquem vértices que satisfaçam as condicionais das linhas 5 e 9 (Algoritmos 13 e 14). Essa contagem é utilizada para compor uma das métricas discutidas no Capítulo 6 que trata do experimento;

- Nas versões implementadas, o conjunto que memoriza os vértices a serem deslocados é chamado de Z e não de U como na versão original.

Algoritmo 13: *Steepest Ascent Direction* (SAD) - *Forward.*

```
1  function SAD()   {ST(V_st, E_st) is a global structure}
2      Z ← ∅; V ← V_st
3      ∀ i ∈ V do C(i) ← i; φ_i ← −α c_i e^{−α^{(S_i+d_i)}}
4      while V ≠ {1} do
5          if (V has a node source i ≠ 1) & (at most one successor j) then
6              iteration_SAD = iteration_SAD + 1
7              C(j) ← C(j) + C(i); φ_j ← φ_j + φ_i; V ← V − i
8          else
9              if (V has a node sink j ≠ 1) & (only one predeccessor i) then
10                 iteration_SAD = iteration_SAD + 1
11                 if φ_j > 0 then Z ← Z + C(j)
12                 else φ_i ← φ_i + φ_j; C(i) ← C(i) + C(j); V ← V − j
13             end
14         end
15     end
16     return(Z)
```

Algoritmo 14: *Steepest Ascent Direction* (SAD) - *Backward.*

```
1  function SAD()   {ST(V_st, E_st) is a global structure}
2      Z ← ∅; V ← V_st
3      ∀ i ∈ V do C(i) ← i; φ_i ← −α c_i e^{−α^(S_i + d_i)}
4      while V ≠ {1} do
5          if (V has a node sink i ≠ 1) & (at most one predecessor j) then
6              iteration_SAD = iteration_SAD + 1
7              C(j) ← C(j) + C(i); φ_j ← φ_j + φ_i; V ← V − i
8          else
9              if (V has a node source j ≠ 1) & (only one successor i) then
10                 iteration_SAD = iteration_SAD + 1
11                 if φ_j ≤ 0 then Z ← Z + C(j)
12                 else φ_i ← φ_i + φ_j; C(i) ← C(i) + C(j); V ← V − j
13             end
14         end
15     end
16     return(Z)
```

4.2.2 Diferenças em *Vertex Ascent* (VA)

No caso do componente *Vertex Ascent* (VA) de SAAFB, as diferenças entre as abordagens $forward$ e *backward* estão indicadas nas linhas sublinhadas nos pseudocódigos em Algoritmos 15 e 16. Essas linhas têm referências opostas no que se refere às ações de deslocamento dos conjuntos de vértices (entre as versões $forward$ e *backward*).

Sobre as diferenças de conveniência ou estilo entre a versão original (Algoritmo 4) e as versões implementadas (Algoritmos 15 e 16), estão:

- Nas versões implementadas, a árvore geradora (ST) e o grafo com todas a restrições (G) são destacados como estruturas globais;
- Nas versões implementadas, usa-se a mesma notação para a função *Compute* v_{k*l*} (linha 4 em Algoritmos 15 e 16) que as versões originais dos algoritmos RS e HS;
- Nas versões implementadas, os argumentos S e Z para a execução de VA estão explícitos, assim com o retorno através de S.

Algoritmo 15: *Vertex Ascent* (VA) - *Forward.*

```
1  function VA(S, Z)    {ST, G are global structures}
2      ∀ (i, j) ∈ ST | (j ∉ C(i)) & (i ∉ C(j)) : ST ← ST − (i, j)
3      while Z ≠ ∅ do
4          Compute v_{k*l*} = min { S_l − S_k − d_k }
5                                 (k*,l*)∈G
6                                 k*∈Z
7                                 l*∉Z
8          Take the set C(j) ∈ Z where k* is contained
9          ∀ i ∈ C(j) : S_i ← S_i + v_{k*l*}
10         Z ← Z − C(j);  ST ← ST + (k*, l*)
11     end
12     return (S)
```

Algoritmo 16: *Vertex Ascent* (VA) - *Backward.*

```
1  function VA(S, Z)    {ST, G are global structures}
2      ∀ (i, j) ∈ ST | (j ∉ C(i)) & (i ∉ C(j)) : ST ← ST − (i, j)
3      while Z ≠ ∅ do
4          Compute V_{l*k*} = min { S_k − f_l }
5                                 (l*,k*)∈G
6                                 k*∉Z
7                                 l*∈Z
8          Take the set C(j) ∈ Z where k* is contained
9          ∀ i ∈ C(j) : S_i ← S_i − V_{l*k*}
10         Z ← Z − C(j);  ST ← ST + (l*, k*)
11     end
12     return (S)
```

4.2.3 Diferenças em *Steepest Ascent Procedure* (SAP)

No caso do componente *Steepest Ascent Procedure* (SAP), as versões $forward$ e *backward* estão indicadas nos pseudocódigos em Algoritmos 17 e 18. A diferença importante entre $forward$ e *backward* está destacada na linha 2 de cada versão. Ou seja, enquanto $forward$ trabalha com um agendamento cedo para as atividades da árvore geradora, *backward* trabalha com um agendamento tarde.

Sobre diferenças de conveniência ou estilo entre as versões implementadas (Algoritmos 17 e 18) e a versão original (Algoritmo 5), estão:

- Nas versões implementadas, a árvore geradora é apresentada como estrutura global (linha 1);
- Nas versões implementadas, a estrutura ES da versão original foi eliminada, uma vez que seu conteúdo é copiado para a estrutura S (linha 2 em Algoritmo 5);
- Nas versões implementadas, o laço da versão original com **Repeat** (Algoritmo 5) foi trocado por um **while** (Algoritmos 17 e 18), que coloca em evidência o critério de parada;

- Nas versões implementadas, antes do **while**, uma primeira busca é realizada com a chamada de *Vertex Ascent* (VA). Assim, o resultado é utilizado como critério de entrada nesse laço;
- Nas versões implementadas, a variável $total\_SAD$ é utilizada para contar o número de reinícios de busca na árvore geradora. Essa contagem também é uma métrica discutida no Capítulo 6 que trata do experimento;
- Nas versões implementadas, a árvore geradora é referida como ST (*Spanning Tree*, assim com no algoritmo *Hybrid Search*) e o grafo com todas as restrições de precedência do projeto (incluindo a aresta extra entre os *dummies*) é referido como grafo G. Por outro lado, a versão original de SAP denomina a árvore geradora como G e o grafo com todas as restrições como N.

Algoritmo 17: *Steepest Ascent Procedure* (SAP) - *Forward.*

```
1  procedure SAP()    {ST is a global structure}
2      S, ST ← Determine the Early Schedule (S) as a vector and a corresponding initial Spanning Tree (ST)
         through the original graph G.
3      total_SAD = 0
4      Z ← SAD()
5      while Z ≠ ∅ do
6          total_SAD = total_SAD + 1
7          S ← VA(S, Z)
8          Z ← SAD()
9      end
10     Report the optimal solution S
```

Algoritmo 18: *Steepest Ascent Procedure* (SAP) - *Backward.*

```
1  procedure SAP()    {ST is a global structure}
2      S, ST ← Determine the Late Schedule (S) as a vector and a corresponding initial Spanning Tree (ST)
         through the original graph G.
3      total_SAD = 0
4      Z ← SAD()
5      while Z ≠ ∅ do
6          total_SAD = total_SAD + 1
7          S ← VA(S, Z)
8          Z ← SAD()
9      end
10     Report the optimal solution S
```

4.3 *HYBRID SEARCH*

O terceiro e último algoritmo implementado para o experimento foi *Hybrid Search* (HS). Como mencionado, a versão original de HS já inclui a capacidade de inversão de busca e deslocamento. Assim, os pseudocódigos correspondentes aos modos $forward$ e *backward* também seguem destacados por componente.

4.3.1 Diferenças em *Recursion* de HS

No caso do componente *Recursion* de HS, as diferenças entre as versões $forward$ e *backward* estão indicadas nos pseudocódigos em Algoritmos 19 e 20. No caso, as linhas 4, 6, 9 e 12 têm ações opostas no que se refere à busca por subárvores.

Sobre diferenças de conveniência ou estilo entre as versões implementadas de *Recursion* (Algoritmos 19 e 20) para a versão original (Algoritmo 6), destacam-se:

- Nas versões implementadas, CA, ST e SS são indicadas como estruturas globais;
- Nas versões implementadas, existe retorno explícito através de SA e DC;
- Nas versões implementadas, a variável $total\_Recursion$ (linha 2) foi incluída e é utilizada para contar o número de chamadas recursivas. Sua inicialização ocorre uma única vez com o valor zero imediatamente antes da primeira chamada ao algoritmo componente $Step\_3$. Essa contagem é utilizada para compor uma das métricas discutidas no Capítulo 6 que trata do experimento.

Algoritmo 19: *Recursion* de HS - $Forward$.

```
1  function Recursion(newnode)    {CA, ST, SS are global structures}
2      total_Recursion = total_Recursion + 1
3      SA ← {newnode}; DC ← DC_newnode; CA ← CA + newnode
4      for (i|i ∉ CA and i succeeds newnode ∈ ST) do
5          SA', DC' ← Recursion(i)
6          if DC' ≥ 0 then
7              SA ← SA + SA'; DC ← DC + DC'
8          else
9              ST ← ST − (newnode, i); SS ← SS + SA'
10         end
11     end
12     for (i|i ∉ CA and i precedes newnode ∈ ST) do
13         SA', DC' ← Recursion(i)
14         SA ← SA + SA'; DC ← DC + DC'
15     end
16     return (SA, DC)
```

Algoritmo 20: *Recursion* de HS - *Backward.*

```
1  function Recursion(newnode)   {CA, ST, SS are global structures}
2      total_Recursion = total_Recursion + 1
3      SA ← {newnode}; DC ← DC_newnode; CA ← CA + newnode
4      for (i|i ∉ CA and i precedes newnode ∈ ST) do
5          SA', DC' ← Recursion(i)
6          if DC' < 0 then
7              SA ← SA + SA'; DC ← DC + DC'
8          else
9              ST ← ST − (i, newnode); SS ← SS + SA'
10         end
11     end
12     for (i|i ∉ CA and i succeeds newnode ∈ ST) do
13         SA', DC' ← Recursion(i)
14         SA ← SA + SA'; DC ← DC + DC'
15     end
16     return (SA, DC)
```

4.3.2 Diferenças em *Shift_activities*

No caso do componente *Shift_activities* de HS, as diferenças entre as versões *forward* e *backward* estão indicadas nos pseudocódigos em Algoritmos 21 e 22. No caso, as linhas sublinhadas 4, 5, 6, 7, 8 e 9 de cada uma das versões *forward* e *backward* realizam ações opostas entre si no que se refere ao deslocamento das subárvores.

Sobre diferenças de conveniência ou estilo entre as versões implementadas de *Shift_activities* (Algoritmos 21 e 22) e a versão original (Algoritmo 7), destaque apenas para os elementos apontados como globais SS, ST e G nas versões implementadas (linha 1).

Algoritmo 21: *Shift_activities - Forward.*

```
1  procedure Shift_activities()   {SS, ST, and G are global structures}
2      Z ← ∅; ∀ i ∈ SA | SA ∈ SS: Z ← Z + i
3      while Z ≠ ∅ do
4          Compute v_{k*l*} = min { s_l − s_k − d_k }
5                                 (k*,l*)∈G;
6                                 k*∈Z;
7                                 l*∉Z
8          ∀ i ∈ SA|k* ∈ SA : s_i ← s_i + v_{k*l*} and Z ← Z − i
9          ST ← ST + (k*, l*)
10     end
```

Algoritmo 22: *Shift_activities - Backward.*

```
1  procedure Shift_activities()   {SS, ST, and G are global structures}
2      Z ← ∅; ∀ i ∈ SA | SA ∈ SS: Z ← Z + i
3      while Z ≠ ∅ do
4          Compute v_{l*k*} = min { s_k − f_l }
5                                 (l*,k*)∈G;
6                                 k*∉Z;
7                                 l*∈Z
8          ∀ i ∈ SA|k* ∈ SA : s_i ← s_i − v_{l*k*} and Z ← Z − i
9          ST ← ST + (l*, k*)
10     end
```

4.3.3 Diferenças em *Hybrid Recursive Search* (HRS)

No caso do componente *Hybrid Recursive Search* (HRS) de HS, as diferenças entre as versões $forward$ e *backward* estão indicadas nos pseudocódigos em Algoritmos 23 e 24. No caso, a linha 4 de cada uma das versões $forward$ e *backward* indica a única diferença, que se refere ao ponto de início da busca (primeiro ou último vértice).

Sobre diferenças de conveniência ou estilo das versões implementadas de HRS (Algoritmos 23 e 24) para a versão original (Algoritmo 8), destacam-se:

- Nas versões implementadas, a indicação de CA e SS como estruturas globais.
- Nas versões implementadas, a troca de um laço com **Repeat** da versão original (Algoritmo 8) por uma abordagem recursiva de HRS (Algoritmos 23 e 24).
- Nas versões implementadas, a variável $total\_HRS$ foi incluída para contar o reinício de buscas na árvore geradora. Essa variável é global e iniciada uma única vez imediatamente antes da primeira chamada ao algoritmo componente HRS. Tal contagem se refere a uma das métricas discutidas no Capítulo 6 que trata do experimento.

Algoritmo 23: *Hybrid Recursive Search* (HRS) - *Forward.*

```
1  procedure HRS()   {CA and SS are global structures}
2      total_HRS = total_HRS + 1
3      CA ← SS ← ∅
4      SA, DC' ← Recursion(1)
5      if SS ≠ ∅ then
6        Shift_activities()
7        HRS()
8      else Report the optimal solution DC'
```

Algoritmo 24: *Hybrid Recursive Search* (HRS) - *Backward.*

```
1  procedure HRS()   {CA and SS are global structures}
2      total_HRS = total_HRS + 1
3      CA ← SS ← ∅
4      SA, DC' ← Recursion(n)
5      if SS ≠ ∅ then
6        Shift_activities()
7        HRS()
8      else Report the optimal solution DC'
```

4.4 FUNÇÃO DE CÁLCULO DE DISTÂNCIA

Compute v_{k*l*} se refere a uma função contida tanto nos algoritmos originais como nas variações implementadas para o experimento desta tese. Como já foi anunciado, no Capítulo 3 que trata do referencial teórico, seu propósito é identificar a menor distância entre um vértice $(k*)$ do conjunto que deve ser deslocado para um vértice $(l*)$ fora desse mesmo conjunto (estratégia $forward$), representando a menor distância entre quem está dentro e fora. *Compute* v_{k*l*} está no cerne da lógica de deslocamento, por isso, faz parte de todos os algoritmos. Mesmo nessa condição, os algoritmos originais (RS, SAA e HS) trataram *Compute* v_{k*l*} como caixa preta. Em outras palavras, nenhum dos trabalhos originais apresentou pseudocódigo para essa função.

Algoritmo 25: *Compute* v_{k*l*} - *Forward.*

```
1  function Compute v_{k*l*}(Z)
2      v_{k*l*} ← δ
3      k ← ∅; l ← ∅
4      for node ∈ Z do
5          if k = ∅ then k ← node
6          for suc ∈ successors of node do
7              edge_checked ← edge_checked + 1
8              if suc ∉ Z do
9                  if s_l − s_k < 0 then
10                     current_min = s_l − s_k − (−δ)
11                 else
12                     current_min = s_l − s_k − d_k
13                 end
14                 if current_min < v_{k*l*} then
15                     v_{k*l*} ← current_min
16                     if node is the last node then
17                         l ← 1
18                     else
19                         l ← suc
20                     end
21                     k ← node
22                 else
23                     if l = ∅ then l ← suc
24                 end
25     return (k, l, v_{k*l*})
```

Assim, nesta seção, as versões implementadas da referida função nos modos $forward$ e *backward* junto aos algoritmos do experimento (RSFB, SAAFB e HS) seguem explicitamente em Algoritmos 25 e 26. Então, cabe destacar alguns dos termos utilizados:

- Z denota o conjunto de vértices que deve ser deslocado;
- δ denota o *deadline* do projeto, usado para iniciar o valor da variável v_{k*l*} ou v_{l*k*} que controla as menores distâncias identificadas;
- s_k e s_l denotam as datas de início dos vértices k e l;
- d_k denota a duração do vértice k;
- f_l denota o fim do vértice l.

Além disso, *edge_checked* se refere a uma variável global de contagem das arestas verificadas, útil para uma das métricas discutidas no Capítulo 6 que trata do experimento. Essa variável é iniciada uma única vez com valor zero, quando qualquer um dos algoritmos também é iniciado (RSFB, SAAFB e HS). Com isso, a função *Compute* retorna o par de vértices $(k, l$ ou $l, k)$ que expressa a aresta de menor distância e o passo de deslocamento a ser incrementado (se for $forward$) ou decrementado (se for *backward*) nas atividades que serão deslocadas. As linhas sublinhadas em Algoritmos 25 e 26 destacam as diferenças entre as abordagens $forward$ e *backward*.

Algoritmo 26: *Compute* v_{l*k*} - *Backward*.

```
1  function Compute v_{l*k*}(Z)
2      v_{l*k*} ← δ
3      k ← ∅; l ← ∅
4      for node ∈ Z do
5          if k = ∅ then k ← node
6          for pred ∈ predecessors of node do
7              edge_checked ← edge_checked + 1
8              if pred ∉ Z do
9                  if s_k − s_l < 0 then
10                     current_min = s_k − f_l − (−δ)
11                 else
12                     current_min = s_k − f_l
13                 end
14                 if current_min < v_{l*k*} then
15                     v_{l*k*} ← current_min
16                     k ← node
17                     l ← pred
18                     if v_{l*k*} < 0 then v_{l*k*} = v_{l*k*} * −1
19                 else
20                     if l = ∅ then l ← suc
21                 end
22     return (k, l, v_{l*k*})
```

5 METODOLOGIA

Neste capítulo se apresenta o tipo de pesquisa utilizado, o método de abordagem empregado, assim como as características consideradas para a verificação das hipóteses.

5.1 TIPO DA PESQUISA

Para Wazlawick (2009) uma pesquisa em ciência da computação pode ser classificada quanto à *natureza*, quanto aos *objetivos* e quanto aos *procedimentos técnicos*. Sob tal proposta, esta tese pode ser classificada conforme os pontos abaixo.

Quanto à *natureza* é:

- *Pesquisa original*, pois apresenta novo conhecimento sobre comportamento de algoritmos de escalonamento de projetos, apoiado em experimento, análises e teorias.

Quanto aos *objetivos* é:

- *Pesquisa exploratória*, pois examina algoritmos de escalonamento de projetos com a finalidade de identificar lacunas teóricas;
- *Pesquisa descritiva*, pois as estruturas e funcionamento dos algoritmos de interesse são descritos detalhadamente;
- *Pesquisa explicativa*, pois busca esclarecer influências de fatores estruturais de redes de projeto sobre os resultados dos escalonamentos gerados pelos algoritmos.

Quanto aos *procedimentos técnicos* é:

- *Pesquisa bibliográfica*, pois é apoiada em artigos, teses e livros;
- *Pesquisa experimental*, pois registra manipulações de variáveis independentes com avaliações sobre os resultados de escalonamento dos algoritmos de interesse, através de experimento.

5.2 MÉTODO DE ABORDAGEM

O método hipotético-dedutivo de Popper (2004) se refere à abordagem metodológica desta pesquisa. A síntese dessa abordagem assume que, a partir de teorias previamente

definidas, rupturas teóricas (lacunas ou problemas) podem ser identificadas. Para tais rupturas são propostas teorias-tentativa (hipóteses) que devem ser passíveis de teste. A etapa de realização dos testes se refere ao falseamento, ou seja, a tentativa de refutação das hipóteses. Se as hipóteses resistirem aos testes de falseamento serão consideradas corroboradas, caso contrário serão consideradas refutadas. Para Popper (2004) esse é um processo com poder de renovar a si mesmo, pois a refutação ou corroboração de hipóteses é capaz de originar novos problemas de pesquisa e novas hipóteses. Nesse sentido, Popper (2004) afirma que a ciência começa com problemas e termina gerando novos problemas.

A Figura 21 apresenta genericamente a abordagem do método hipotético-dedutivo em onze etapas. No caso, deve ser observado que, através de refutação ou corroboração de hipóteses, o fluxo é cíclico com o surgimento de novos problemas com possíveis novas hipóteses.

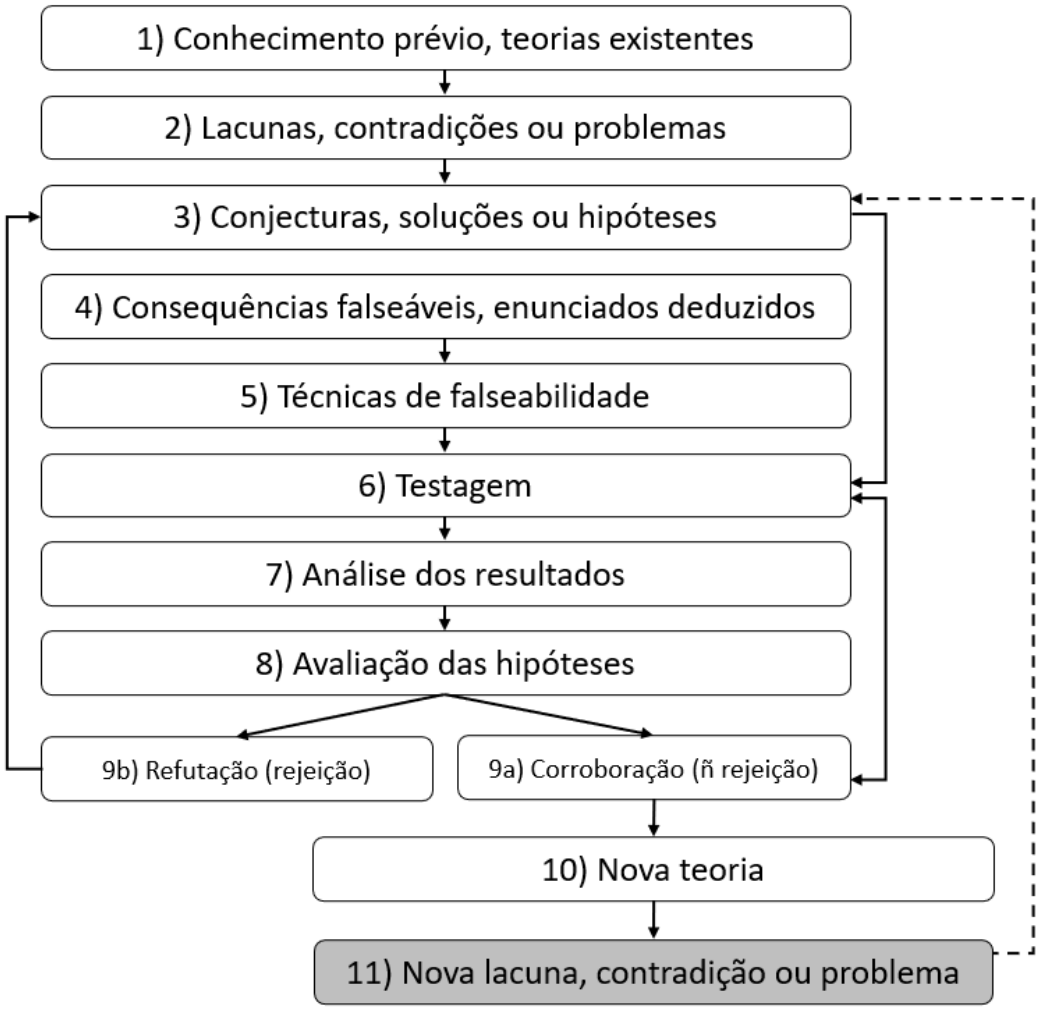

Figura 21 – Diagrama genérico do método hipotético-dedutivo.

A Figura 22 mostra o mesmo diagrama do método, mas adaptado ao contexto e instrumentos desta tese. Na etapa 1, o contexto teórico inclui problemas e algoritmos de escalonamento de projetos para *max-npv*. Na etapa 2, as principais lacunas (problemas) se referem à complexidade aberta dos algoritmos de interesse. Na etapa 3, as três hipóteses da pesquisa são tomadas como solução provisória. Na etapa 4, destacam-se as consequências falseáveis. Na etapa 5, as hipóteses nulas em oposição às hipóteses alternativas são definidas. Na etapa 6, destaca-se que um experimento fatorial é instrumento de testagem. Na etapa 7, destaca-se que a análise dos resultados do experimento faz uso dos métodos de Kolmogorov-Smirnov e Modelos de Regressão. Na etapa 8, as hipóteses são avaliadas à luz dos resultados. Na etapa 9, destacam-se as hipóteses corroboradas e ou refutadas. Na etapa 10, um novo conhecimento teórico sobre o comportamento dos algoritmos é obtido. Na etapa 11, tem-se a oportunidade de identificar novas lacunas, que podem renovar o ciclo com novas hipóteses.

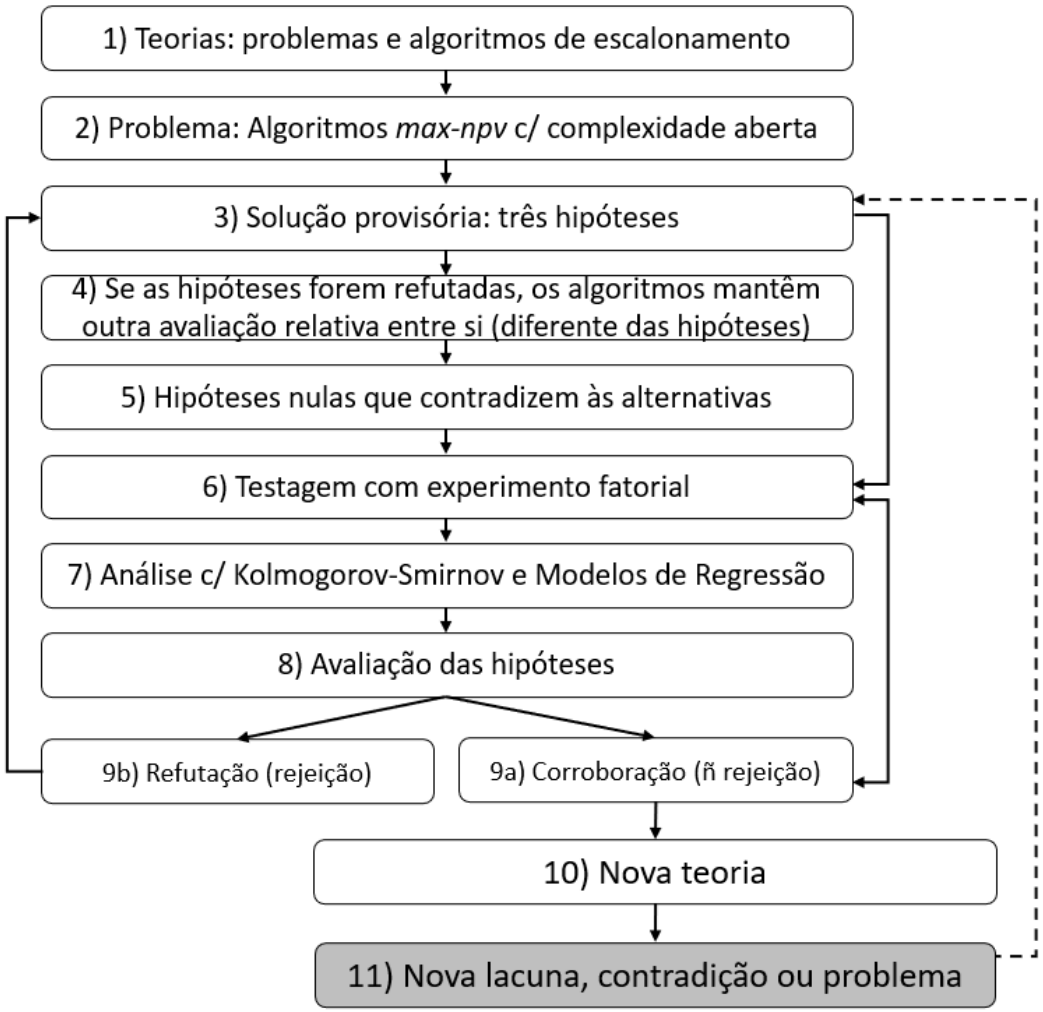

Figura 22 – Diagrama parcialmente adaptado do método hipotético-dedutivo.

5.3 VERIFICAÇÃO DAS HIPÓTESES

Considerando que as hipóteses devem ser testadas (sendo refutadas ou corroboradas), deve-se definir quais serão as hipóteses alternativas e quais serão as hipóteses nulas. Nesse sentido, as hipóteses já apresentadas no Capítulo 1 serão consideradas como as hipóteses alternativas. Com isso, as hipóteses nulas serão oposições às alternativas. Portanto, se as hipóteses nulas (H_0) forem refutadas, as hipóteses alternativas (H_a) serão corroboradas. Sendo assim, as hipóteses nulas podem ser descritas da seguinte maneira:

- **Hipótese I_0 ($H1_0$):**
  O algoritmo RSFB **não está** em ordem (classe) de complexidade global maior que os algoritmos SAAFB e HS.

- **Hipótese II_0 ($H2_0$):**
  Os algoritmos HS e SAAFB **não estão** na mesma ordem (classe) de complexidade global.

- **Hipótese III_0 ($H3_0$):**
  Os algoritmos RSFB, SAAFB e HS **não estão** na mesma ordem (classe) de complexidade, no que se refere ao número de buscas nas respectivas árvores geradoras.

5.4 TRATAMENTO E ANÁLISE DOS DADOS

Embora a caracterização dos dados utilizados, assim como dos instrumentos de tratamento e análise pudessem fazer parte deste capítulo, optou-se em apresentá-los no próximo capítulo que descreve integralmente o experimento da tese.

6 EXPERIMENTO

Este capítulo descreve as variáveis utilizadas (independentes e dependentes), as três amostras de dados empregadas, a modelagem linear generalizada (como instrumento estatístico) e os resultados produzidos pelos algoritmos em 14.000 instâncias, com análise preliminar e apresentação de vinte oito modelos estatísticos preditores.

6.1 VARIÁVEIS INDEPENDENTES: FATORES

O experimento teve como objetivo investigar o desempenho computacional absoluto e relativo dos algoritmos RSFB, SAAFB e HS sob diferentes características de redes de projeto. Essas características, que também podem ser chamadas de fatores experimentais, constituem as variáveis independentes do experimento. A Tabela 2 mostra os fatores experimentais e seus respectivos significados.

Tabela 2 – Fatores experimentais.

Fator	**Código**	**Descrição**
vertices	f_1	Número de vértices do grafo.
layers	f_2	Número de camadas do grafo.
maxDegree	f_3	Grau máximo (*in* e *out*) dos vértices do grafo.
discRate(%)	f_4	Taxa de desconto usada no projeto.
percNeg(%)	f_5	Percentual de atividades com fluxo de caixa negativo.
cpMult	f_6	*Deadline* como um múltiplo da duração do caminho crítico.
edges	f_7	Número de arestas do grafo.

6.2 VARIÁVEIS DEPENDENTES: MÉTRICAS DIRETAS

As variáveis dependentes do experimento foram selecionadas de forma a fornecer uma medida do custo computacional dos algoritmos de interesse. Essas variáveis foram definidas por duas métricas. A primeira métrica, denominada *computational cost*, representa o custo computacional total realizado pelos algoritmos para avaliar os escalonamentos ótimos. Assim, a métrica *computational cost* é obtida somando-se o número de iterações (nos algoritmos iterativos) ou o número de chamadas recursivas (nos algoritmos recursivos) ao número de arestas verificadas na busca pela menor distância de deslocamento das subárvores. As duas expressões que correspondem à métrica *computational cost* (expressão 6.1 para algoritmos recursivos e expressão 6.2 para iterativos) seguem exibidas abaixo:

$$computational\ cost = total\_Recursion + edge\_checked \tag{6.1}$$

$$computational\ cost = iteration\_SAD + edge\_checked \tag{6.2}$$

Nos algoritmos RSFB e HS, a composição de *computational cost* se refere ao indicado na expressão 6.1. Nesse caso, a variável $total\_Recursion$ (linha 2 dos pseudocódigos em Algoritmos 11 e 19) conta o número de chamadas recursivas e a variável $edge\_checked$ (a linha 7 do pseudocódigo em Algoritmo 25) conta as arestas verificadas. No algoritmo SAAFB, a composição de *computational cost* se refere ao indicado na expressão 6.2. Neste caso, a variável $iteration\_SAD$ (linhas 6 e 10 do pseudocódigo em Algoritmo 13) conta as iterações realizadas e a variável $edge\_checked$ (também na linha 7 do pseudocódigo em Algoritmo 25) conta as arestas verificadas.

A segunda métrica, chamada *restarted search*, representa o número de vezes que uma nova pesquisa é reiniciada na árvore geradora. No caso do algoritmo RSFB, *restarted search* é medida pela variável $total\_Step\_3$, indicada na linha 2 do pseudocódigo em Algoritmo 9. Essa variável guarda a contagem de todas as chamadas para o componente $Step\_3$ de RSFB. No caso do algoritmo SAAFB, *restarted search* é medida pela variável $total\_SAD$, indicada na linha 3 do pseudocódigo em Algoritmo 17. Essa variável guarda a contagem de todas as chamadas ao componente SAD de SAAFB. No caso do algoritmo HS, *restarted search* é medida pela variável $total\_HRS$, indicada na linha 2 do pseudocódigo no Algoritmo 23. Essa variável contabiliza todas as chamadas para o componente HRS de HS.

6.3 VARIÁVEIS DEPENDENTES: MÉTRICA DE LIMITE SUPERIOR

As medidas computacionais mais comuns de custo de algoritmo são a) limite superior assintótico (O), b) limite inferior assintótico (Ω), c) limite superior e inferior assintótico (Θ). Nesse sentido, este experimento se preocupou em encontrar um limite superior estatístico assintótico (O) na forma de uma função de custo máximo empírico como um retorno aproximado do custo computacional máximo em função de um fator experimental, ou seja, $maxCost(fator = valor_{fator})$. Nesse caso, a função retorna o custo computacional máximo obtido no experimento quando o $fator$ específico assume $valor_{fator}$.

Como cada um dos sete fatores experimentais (f_i) assume valores em um conjunto discreto Df_i, todo o espaço amostral do experimento é dado por:

$$S = \prod_{i=1}^{i=7} Df_i$$

Assim, o espaço amostral do experimento é a relação que contém todas as tuplas que podem ser formadas com todas as combinações possíveis de valores de fatores experimentais, como segue:

$$S = \{(vf_1, vf_2, \ldots, vf_n) | (vf_1 \in Df_1, vf_2 \in Df_2, \ldots, vf_n \in Df_n)\}$$

.

Desse modo, no espaço amostral, a função de custo computacional é $S \rightarrow R^+$ e um subespaço amostral é $S_{f_i,vf_i} = \{s \in S | s(\ldots, f_i = vf_i, \ldots\}$, que é subconjunto de S, contendo todas as tuplas onde o fator f_i assume o valor vf_i. Da mesma forma, a função de custo computacional empírico é $Cost(vf_1, vf_2, \ldots, vf_n) = Cost(f_1 = vf_1, f_2 = vf_2, \ldots, f_n = vf_n)$, que é o valor experimental obtido para cada elemento do espaço amostral S.

Por fim, a função empírica $maxCost(f_i, vf_i)$ pode ser definida como:

$$maxCost(f_i, vf_i) = max(Cost(s_i | s_i \in S_{f_i,vf_i}))$$

.

6.4 CONJUNTO DE DADOS: CARACTERÍSTICAS DAS AMOSTRAS

O experimento executou os algoritmos em três amostras compreendendo 14.000 instâncias de rede de projeto (5.000 na Amostra 1, 5.000 na Amostra 2 e 4.000 na Amostra 3). As instâncias foram criadas a partir de um gerador de redes aleatórias, desenvolvido exclusivamente para o experimento, inspirado em Erdos, Rényi et al. (1960). Vale destacar que o tempo para gerar as instâncias da Amostra 3 foi muito maior do que o tempo para gerar as instâncias das Amostras 1 e 2 (medido em segundos). Desse modo, por questão de conveniência, o número de instâncias da Amostra 3 é um pouco menor que o número das demais. Assim, o gerador foi utilizado para criar instâncias aleatórias a partir de intervalos de valores para os parâmetros conforme a Tabela 3.

Tabela 3 – Amostragem aleatória.

Parâmetro	**Amostra 1**	**Amostra 2**	**Amostra 3**
vertices	16..80	16..320	16..320
layers	2..*vertices* − 1	2..*vertices* − 1	2
maxDegree	2 ou 3	2 ou 3	(*vertices* − 2)/2
discRate(%)	1..20	1..20	1..20
percNeg(%)	0,10,20,..100	0,10,20,..100	0,5,10,15...50
cpMult	1..2	1..2	1..2
cashFlow	-100..100	-100..100	-100..100
activityDur	5..10	5..10	5..10

As Amostras 1 e 2 diferem entre si apenas no número de vértices das redes, por conta da intenção de avaliar o comportamento dos algoritmos em dois conjuntos distintos de redes de projeto: redes pequenas/médias (Amostra 1) e redes grandes (Amostra 2).

A Amostra 3, por sua vez, foi criada para avaliar o comportamento dos algoritmos em grafos com apenas duas camadas (desconsiderando *dummies*), em que todos os vértices da primeira camada estão conectados a todos os vértices da segunda camada. Tal configuração gerou grafos com o número máximo de arestas, ou seja, dígrafos bipartidos completos (desconsiderando *dummies*). Isso forneceu uma amostra de dados desenhada para estressar os algoritmos com um alto número de arestas na busca por subárvores candidatas a deslocamento. Ainda na Amostra 3, o parâmetro *percNeg* utilizou um intervalo com valores espaçados de 5 em 5 e limitado a 50. Tal espaçamento foi utilizado com o intuito de suavizar os modelos estatísticos para esse parâmetro. Já o limite de 50, também para o parâmetro *percNeg*, foi utilizado por se tratar de grafos com apenas duas camadas, fazendo com que no máximo uma das camadas tenha integralmente atividades com fluxo de caixa negativo.

Os algoritmos foram codificados em *Python 3* e executados em um computador pessoal, com processador *core i5* de 2,50 GHz, com 32 GB de RAM, rodando em *Windows* 10. Vale ressaltar que embora existam instâncias disponíveis na literatura, optou–se por criar e utilizar um gerador específico para este experimento com a finalidade de estabelecer faixas de valores de parâmetros com aumento gradual.

6.5 INSTRUMENTO DE MODELAGEM ESTATÍSTICA: MLG

A regressão linear clássica foi o primeiro instrumento de análise de dados considerado. No entanto, como alguns dos pressupostos da abordagem de regressão linear clássica (normalidade dos resíduos e homocedasticidade) não foram satisfeitos, mesmo com transformações de Box-Cox e logarítmicas, outra ferramenta foi utilizada: a regressão

generalizada. Essa abordagem faz parte do grupo denominado Modelos Lineares Generalizados (MLG), do inglês *Generalized Linear Models* (GLM), grupo introduzido por Nelder e Wedderburn (1972). Nesse caso, a abordagem de regressão generalizada admite modelos lineares e também não lineares, além de variáveis dependentes com distribuições como Bernoulli, Poisson, Poisson-Gamma e a própria Gaussiana (normal).

Segundo Fávero e Belfiore (2019), a definição adequada de um modelo linear generalizado deve considerar a característica da variável dependente. Nesse sentido, merecem destaque as características de dois tipos de variáveis dependentes apresentadas por Fávero e Belfiore (2019). Na primeira, o modelo generalizado é denominado de tipo linear quando a variável dependente é quantitativa e aderente à distribuição normal. Na segunda, quando a variável dependente é quantitativa, sendo dado de contagem (valores inteiros e não negativos), não aderente à distribuição Gaussiana, deve-se considerar a distribuição Poisson ou Poisson-Gamma. Vale ressaltar que a diferença entre uma distribuição Poisson e uma distribuição Poisson-Gamma é a cauda longa à direita da segunda distribuição (Poisson-Gamma), caracterizando uma superdispersão dos dados. Em outras palavras, quando a variância é estatisticamente maior que a média (com dados de contagem), deve-se escolher a distribuição Poisson-Gamma. Nesse caso, o modelo generalizado é chamado de binomial negativa.

Neste trabalho, todos os resultados foram analisados utilizando modelos generalizados com distribuição binomial negativa (Poisson-Gamma).

6.6 RESULTADOS: ANÁLISE PRELIMINAR DOS DADOS

A análise preliminar dos resultados experimentais seguiu os seguintes passos: (1) gerar uma análise sumária dos dados, (2) verificar a similaridade entre as distribuições dos resultados empíricos e (3) verificar o grau de correlação entre os fatores e as variáveis dependentes.

6.6.1 Sumário das variáveis dependentes

As Tabelas 4, 5, 6, 7, 8 e 9 apresentam os resultados de escalonamento produzidos pelos algoritmos em termos de valores mínimo, primeiro quartil, mediana, média, terceiro quartil e máximo, por métrica e tipo de amostra.

Na Tabela 4, referente à métrica *computational cost* na Amostra 1, é possível notar que os valores para os três algoritmos no primeiro quartil e também no segundo quartil (mediana) são muito próximos. No entanto, no terceiro quartil, o valor para o algoritmo

RSFB é quase o dobro dos respectivos valores de SAAFB e HS. Ressalta-se também que o valor máximo de RSFB é cerca de cinco vezes maior que os respectivos valores de SAAFB e HS. Na Tabela, 5, referente à métrica *restarted search* na Amostra 1, a mediana de RSFB é três vezes maior que o valor de SAAFB e HS, assim como a média. No terceiro quartil, o valor de RSFB é cinco vezes maior que os demais. O valor máximo de RSFB é cerca de vinte vezes maior que os dos outros algoritmos.

Tabela 4 – Sumário Amostra 1 - métrica *computational cost*.

Algo.	Mín.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	18	93	183	302	377	4308
RSFB	18	77	262	614	689	25587
SAAFB	17	91	179	295	365	4298

Tabela 5 – Sumário Amostra 1 - métrica *restarted search*.

Algo.	Mín.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	1	2	3	3	4	15
RSFB	1	3	9	15	20	339
SAAFB	1	2	3	3	4	15

As Amostras 2 e 3 têm praticamente os mesmos resultados entre os algoritmos SAAFB e HS, conforme mostram as Tabelas 6, 7, 8 e 9. Neste ponto, vale ressaltar que o algoritmo RSFB apresentou estouro de pilha de recursão em grafos com até 320 vértices. Assim, o uso de RSFB nas Amostras 2 e 3 se tornou inviável. Vale lembrar que as Amostras 2 e 3 possuem grafos com quantidade de vértices que variam de 16 a 320.

Tabela 6 – Sumário Amostra 2 - métrica *computational cost*.

Algo.	Mín.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	18	310	978	2795	3008	58409
SAAFB	17	307	973	2791	3003	58391

Tabela 7 – Sumário Amostra 2 - métrica *restarted search*.

Algo.	Mín.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	1	2	3	4	5	26
SAAFB	1	2	3	4	5	26

Tabela 8 – Sumário Amostra 3 - métrica *computational cost*.

Algo.	Mín.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	18	2706	32826	349093	271009	10186880
SAAFB	17	2631	30819	347432	264012	10186639

Tabela 9 – Sumário Amostra 3 - métrica *restarted search*.

Algo.	Min.	1º Q.	Mediana	Média	3º Q.	Máx.
HS	1	8	21	32	47	241
SAAFB	1	8	21	32	47	241

Os dados das Tabelas 6, 7, 8 e 9 também podem ser avaliados através de gráfico tipo *boxplot*, como mostram as Figuras 23 e 24. No caso da Amostra 1, para as duas métricas, pode-se observar que a cauda longa de RSFB indica maior degeneração quando comparada à cauda de SAAFB e HS. Já nas Amostras 2 e 3, é possível verificar que SAAFB e HS têm a mesma degeneração nas duas métricas. Também é possível constatar que todas as amostras são positivamente assimétricas, ou seja, possuem respectivas medianas mais próximas de seus respectivos primeiros quartis do que de seus respectivos terceiros quartis. Além disso, a quantidade considerável de *outliers* sugere dificuldades com alguns métodos estatísticos paramétricos.

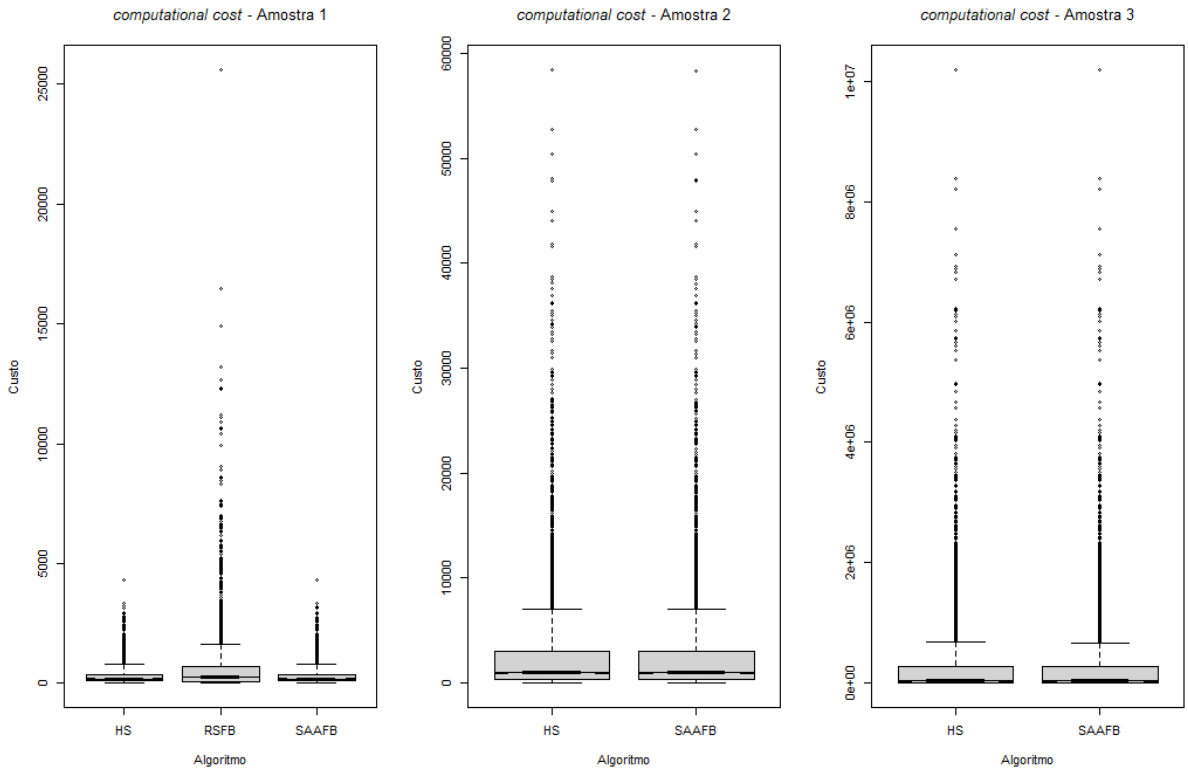

Figura 23 – Sumário da métrica *computational cost* - Amostras 1, 2 e 3.

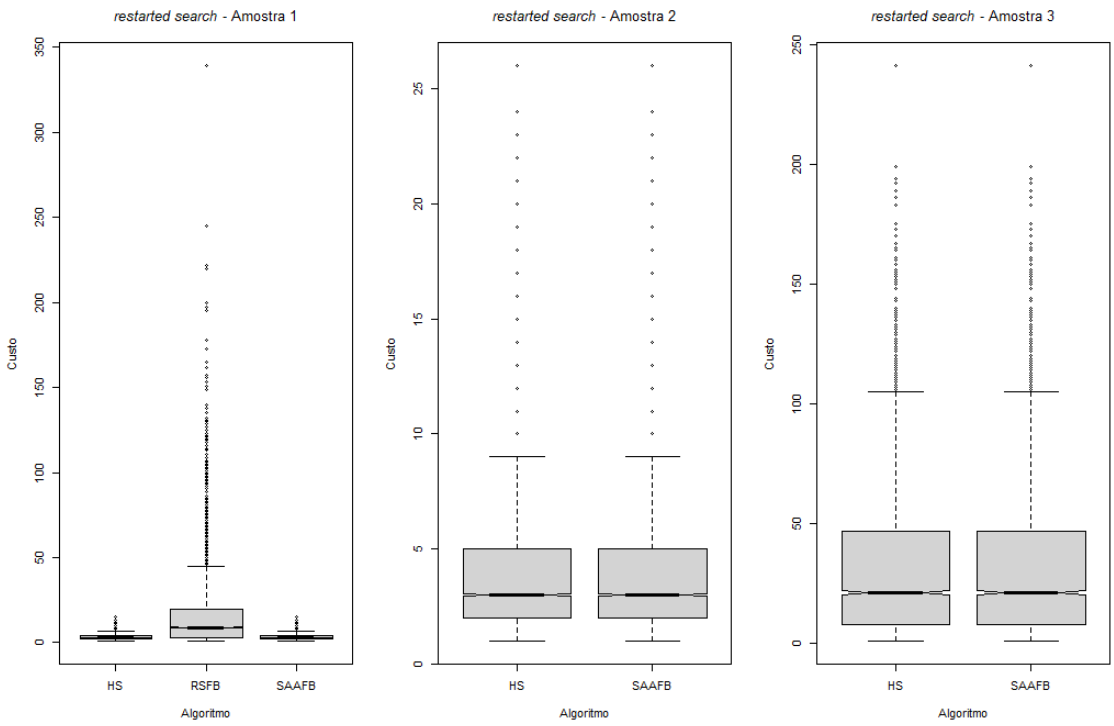

Figura 24 – Sumário da métrica *restarted search* - Amostras 1, 2 e 3.

6.6.2 Tipo de distribuição dos resultados

As distribuições das métricas também foram avaliadas com auxílio de histogramas. Na Amostra 1, conforme indicam as Figuras 25 e 26, é possível notar que as distribuições possuem maior concentração de resultados à esquerda e apresentam queda exponencial com cauda longa à direita. Esse padrão sugere que as distribuições podem ser Poisson ou Poisson-Gamma, conforme argumenta Fávero e Belfiore (2019), considerando ainda que essas variáveis são dados de contagem.

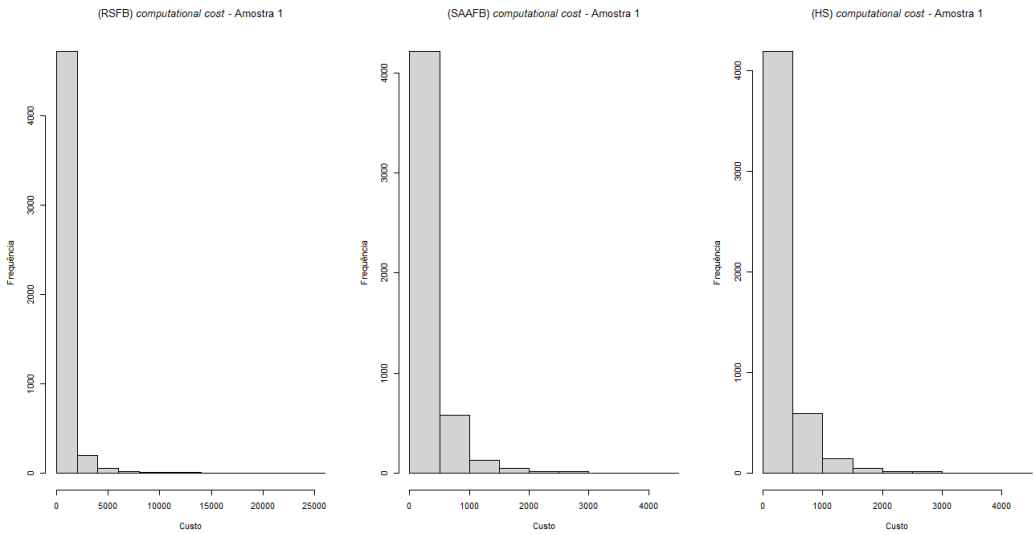

Figura 25 – Distribuições de *computational cost* - Amostra 1.

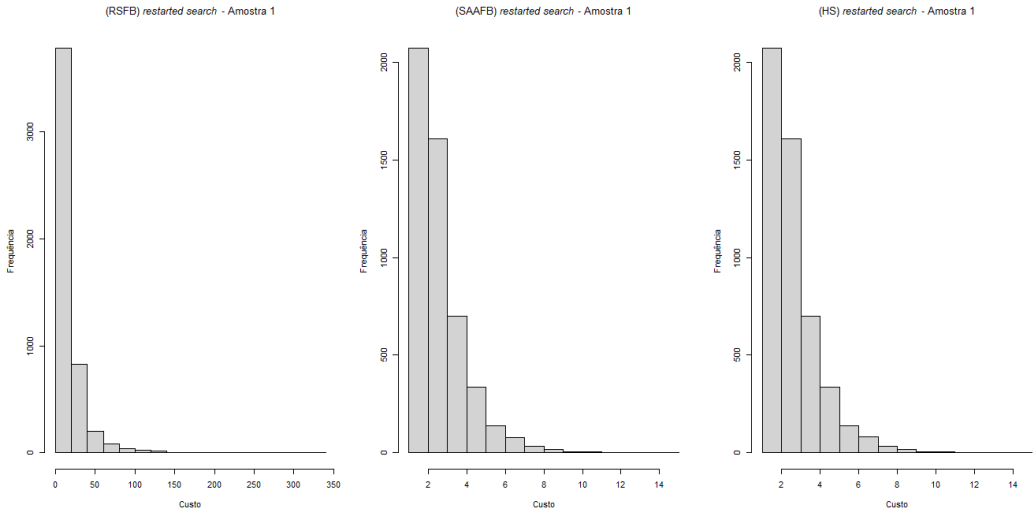

Figura 26 – Distribuições de *restarted search* - Amostra 1.

Nos casos das Amostras 2 e 3, conforme Figuras 27 e 28, também se pode ver o mesmo padrão de concentração de resultados à esquerda e cauda longa à direita. Ao mesmo tempo, para *computational cost*, observa-se que os maiores resultados na Amostra 3 são da ordem de 10.000.000, enquanto na Amostra 2 os maiores resultados são da ordem de 60.000. É importante lembrar que as Amostras 2 e 3 possuem grafos que variam de 16 a 320 vértices, mas que os maiores resultados da Amostra 3 são cerca de 166 vezes maiores que os da Amostra 2 para os mesmos algoritmos (SAAFB e HS). Essa grande diferença na métrica principal está relacionada ao alto número de arestas nos grafos da Amostra 3, que são dígrafos bipartidos completos (desconsiderando *dummies*).

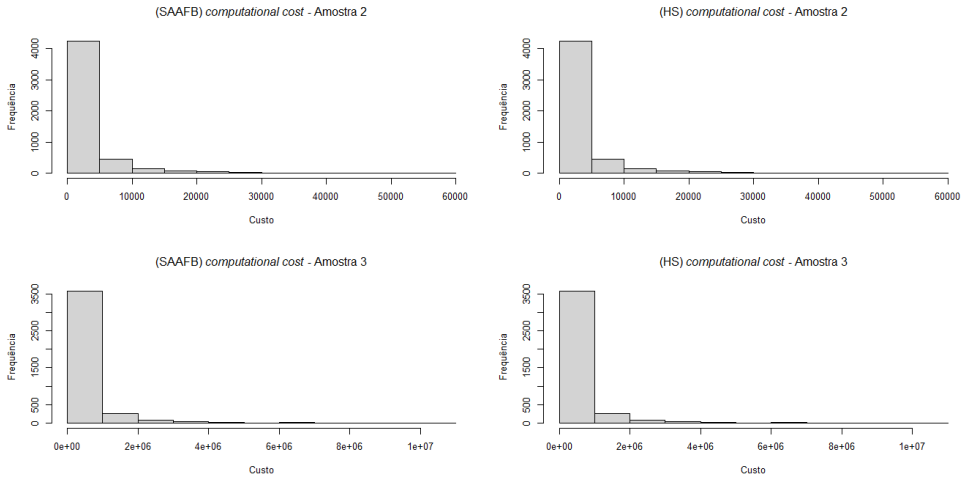

Figura 27 – Distribuições de *computational cost* - Amostras 2 e 3.

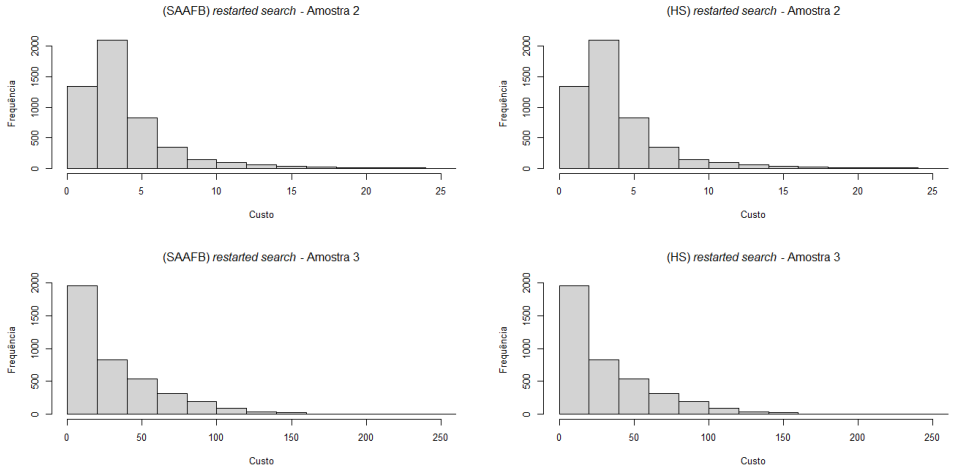

Figura 28 – Distribuições de *restarted search* - Amostras 2 e 3.

6.6.3 Testes de similaridade das distribuições

Como os dados do experimento são não-paramétricos (sem distribuição gaussiana), as distribuições das amostras foram comparadas por meio do teste de Kolmogorov-Smirnov (KS), que permite avaliar a similaridade estatisticamente relevante entre duas distribuições. Nesse caso, as comparações foram feitas por amostra, métrica e algoritmo, conforme Tabelas 10 e 11. Essas tabelas mostram que as distribuições de SAAFB e HS são sempre estatisticamente semelhantes (através das colunas *valor p* e *Similar*). Em contrapartida, as distribuições de RSFB não possuem similaridade estatística com nenhuma das demais.

Tabela 10 – Comparação de distribuições com KS - métrica *computational cost*.

Amostra	Comparação	estatística D	valor p	Similar
1	RSFB *vs* SAAFB	0.17428	2.2e-16	Não
1	RSFB *vs* HS	0.17160	2.2e-16	Não
1	SAAFB *vs* HS	0.01260	0.8222	Sim
2	SAAFB *vs* HS	0.00340	1.0000	Sim
3	SAAFB *vs* HS	0.00775	1.0000	Sim

Tabela 11 – Comparação de distribuições com KS - métrica *restarted search*.

Amostra	Comparação	estatística D	valor p	Similar
1	RSFB *vs* SAAFB	0.57580	2.2e-16	Não
1	RSFB *vs* HS	0.57560	2.2e-16	Não
1	SAAFB *vs* HS	0.00000	1.0000	Sim
2	SAAFB *vs* HS	0.00000	1.0000	Sim
3	SAAFB *vs* HS	0.00125	1.0000	Sim

6.6.4 Correlação entre fatores e métricas

Sobre os fatores experimentais, também foram feitas avaliações quanto à possível correlação com as variáveis dependentes *computational cost* e *restarted search*. Para todas as amostras, o método de correlação utilizado foi o Coeficiente de Spearman, uma vez que as distribuições não foram satisfeitas nos testes de normalidade (os dados são não-paramétricos). A correlação levou em consideração apenas o máximo (de *computational cost* e *restarted search*) para cada valor dos fatores experimentais. No caso do fator f_5 (*percNeg*), em todos os algoritmos, a correlação foi avaliada apenas para valores com até 50% de atividades negativas, pois os algoritmos invertem o sentido da busca quando esse valor é maior do que isso.

Assim, na Amostra 1, de acordo com a Tabela 12, é possível notar que para a métrica *computational cost* os três fatores com os maiores coeficientes de correlação foram f_1 (*vertices*), f_5 (*percNeg*) e f_7 (*edges*). Para a métrica *restarted search* a maior correlação foi com f_1 (*vertices*), f_5 (*percNeg*), f_7 (*edges*) e f_4 (*discRate*). No entanto, a correlação da métrica *restarted search* com *edges* é menor que *computational cost* com *edges*. Os demais fatores não apresentaram coeficientes relevantes. Vale ressaltar que como para os fatores f_3 (*maxDegree*) e f_6 (*cpMult*) os respectivos intervalos de sorteio incluem apenas dois valores possíveis, seus coeficientes foram considerados Não Aplicáveis (NA).

Na Amostra 2, conforme a Tabela 14, é possível notar que os dois fatores com os maiores coeficientes de correlação com a métrica *computational cost* também foram f_1 (*vertices*) e f_5 (*percNeg*), ambos com valores acima de 90%. Os demais fatores não

Tabela 12 – Coeficiente de Spearman (*computational cost vs fatores*) - Amostra 1.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
RSFB	0.93	-0.20	NA	0.67	0.94	NA	0.78
SAAFB	0.96	-0.18	NA	0.70	1.00	NA	0.85
HS	0.96	-0.18	NA	0.70	1.00	NA	0.85

Tabela 13 – Coeficiente de Spearman (*restarted search vs fatores*) - Amostra 1.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
RSFB	0.86	-0.29	NA	0.79	0.94	NA	0.68
SAAFB	0.78	-0.40	NA	0.44	0.71	NA	0.56
HS	0.78	-0.40	NA	0.44	0.71	NA	0.56

apresentaram coeficientes de correlação relevantes. A Tabela 15 mostra que a maior correlação de *restarted search* foi com o fator f_1 (*vertices*).

Tabela 14 – Coeficiente de Spearman (*computational cost vs fatores*) - Amostra 2.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
SAAFB	0.93	-0.13	NA	0.37	1.00	NA	0.76
HS	0.93	-0.13	NA	0.37	1.00	NA	0.75

Tabela 15 – Coeficiente de Spearman (*restarted search vs factors*) - Amostra 2.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
SAAFB	0.70	-0.32	NA	0.58	0.39	NA	0.43
HS	0.70	-0.32	NA	0.58	0.39	NA	0.43

Na Amostra 3, de acordo com as Tabelas 16 e 17, é possível notar que quatro fatores apresentaram fortes coeficientes de correlação: f_1 (*vertices*), f_3 (*maxDegree*), f_5 (*percNeg*) e f_4 (*discRate*). É importante dizer que como o fator *maxDegree*, na Amostra 3, inclui uma faixa de valores de $(16-2)/2$ a $(320-2)/2$ (de acordo com a Tabela 3), o coeficiente de correlação é aplicável e apresenta um valor relevante, diferente das Amostras 1 e 2. Os demais fatores não apresentaram valores de coeficiente de correlação relevantes.

Tabela 16 – Coeficiente de Spearman (*computational cost vs fatores*) - Amostra 3.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
SAAFB	0.93	NA	0.96	0.75	1.00	NA	0.53
HS	0.93	NA	0.96	0.75	1.00	NA	0.53

Tabela 17 – Coeficiente de Spearman (*restarted search vs fatores*) - Amostra 3.

	vertices	*layer*	*maxDegree*	*discRate*	*percNeg*	*cpMult*	*edges*
SAAFB	0.85	NA	0.88	0.88	0.99	NA	0.36
HS	0.85	NA	0.88	0.88	0.99	NA	0.36

Para fechar esta seção, a Tabela 18 mostra a correlação entre a métrica principal *computational cost* (variável dependente) e o tempo de execução dos algoritmos para escalonar os projetos (tempo expresso em milissegundos). Nesse caso, como os algoritmos invertem a direção da busca quando *percNeg* é maior que 50%, a correlação considerada foi limitada aos valores desse fator com até 50%. Como o algoritmo RSFB não foi utilizado nas Amostras 2 e 3, NA se refere a Não é Aplicável. Nos demais casos, há uma correlação forte (acima de 70%) ou muito forte (acima de 90%) entre *computational cost* e a medida de tempo (que por questão de escopo não foi tratada como variável dependente). Não foi feita correlação entre *restarted search* e tempo de execução, pois essa variável dependente não se refere ao custo total, como é o caso de *computational cost*.

Tabela 18 – Coeficiente de Spearman (*computational cost vs* tempo de execução).

	Amostra 1	Amostra 2	Amostra 3
RSFB	0.90	NA	NA
SAAFB	0.79	0.79	0.96
HS	0.86	0.92	0.96

6.7 RESULTADOS: MODELOS ESTATÍSTICOS

Os resultados experimentais foram analisados com a técnica de modelagem estatística denominada MLG, como previamente anunciado. Vinte e oito modelos estatísticos foram gerados e relatados a partir dos resultados das duas métricas (*computational cost* e *restarted search*). Todos os modelos assumiram distribuição Poisson-Gamma (binomial negativa), o que está de acordo com os resultados da análise preliminar. Foram desenvolvidos dois tipos de modelos, com um fator e com dois fatores, considerando a formulação $maxCost$ da Seção 6.3.

6.7.1 Modelos da Amostra 1

Os resultados da Amostra 1 (redes entre 16 e 80 vértices) permitiram a criação de onze modelos: nove referentes à métrica *computational cost* e três referentes à métrica *restarted search*.

6.7.1.1 Modelos para *computational cost* (Amostra 1)

O Modelo 1 (M1) se refere à função $maxCost(vertices)$ para o algoritmo RSFB em função do fator *vertices*. A Figura 29(a) destaca a curva, o polinômio e a proporção do desvio explicado com D^2 ajustado (D^2 *adj.*) do modelo. É importante destacar que em MLG, D^2 ajustado equivale a R^2 ajustado (proporção da variação explicada). A Tabela

19 mostra o respectivo valor estimado, erro padrão, valor da estatística z de Wald e o valor $Pr(>|z|)$. Sobre $Pr(>|z|)$, vale dizer que quando os valores são $< 0,05$ (dentro do nível de significância adotado de 5%), o parâmetro é considerado relevante para o modelo. Assim, no Modelo 1, o polinômio que expressa o comportamento do algoritmo é: $maxCost(x) = 8 + 7x - 1x^2 + 1x^3$, onde x se refere ao parâmetro *vertices*.

Tabela 19 – RSFB $maxCost(vertices)$ para *computational cost* - Amostra 1.

Modelo 1 (M1)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	8.1340	0.0421	193.33	0.0000
poly(x, 3)1	7.2306	0.3395	21.29	0.0000
poly(x, 3)2	-1.2399	0.3395	-3.65	0.0003
poly(x, 3)3	0.8342	0.3395	2.46	0.0140

Tabela 20 – SAAFB $maxCost(vertices)$ para *computational cost* - Amostra 1.

Modelo 2 (M2)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	6.9091	0.0261	264.79	0.0000
poly(x, 2)1	6.4801	0.2115	30.64	0.0000
poly(x, 2)2	-0.9662	0.2112	-4.57	0.0000

Tabela 21 – HS $maxCost(vertices)$ para *computational cost* - Amostra 1.

Modelo 3 (M3)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	6.9329	0.0270	256.52	0.0000
poly(x, 2)1	6.3491	0.2189	29.00	0.0000
poly(x, 2)2	-0.9799	0.2187	-4.48	0.0000

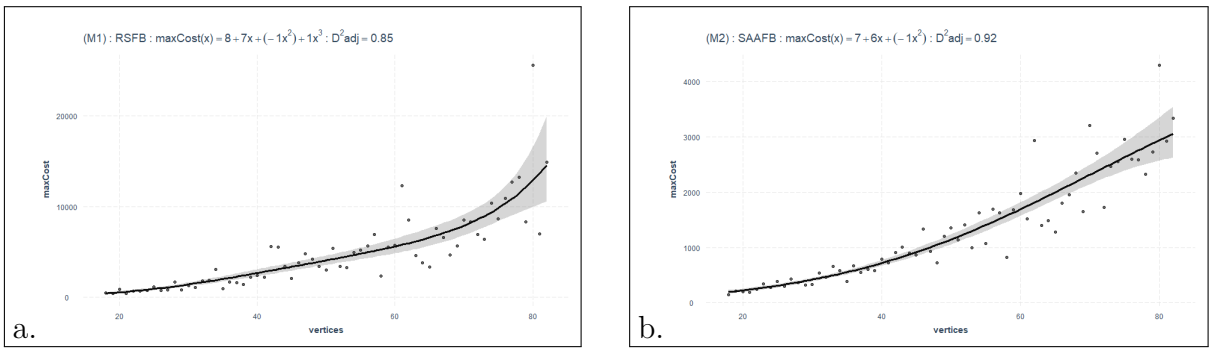

Figura 29 – RSFB e SAAFB $maxCost(vertices)$ para *computational cost* - Amostra 1.

Os Modelos 2 (M2) e 3 (M3) se referem à função $maxCost(vertices)$ dos algoritmos SAAFB e HS. A Figura 29(b) e a Figura 30(a) destacam as respectivas curvas, polinômios e D^2 ajustados dos modelos. As Tabelas 20 e 21 mostram os resultados dos modelos.

Assim, M2 e M3 têm o mesmo polinômio: $maxCost(x) = 7 + 6x - 1x^2$, onde x se refere ao parâmetro *vertices*.

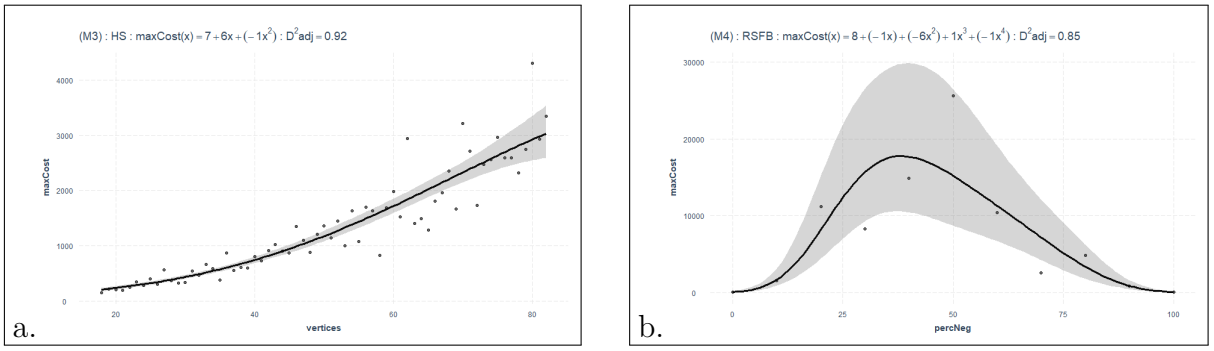

Figura 30 – HS $maxCost(vertices)$, RSFB $maxCost(percNeg)$, *comp. cost* - Amostra 1.

Os Modelos 4 (M4), 5 (M5) e 6 (M6) se referem à função $maxCost(percNeg)$ dos algoritmos RSFB, SAAFB e HS para *computational cost*. As Figuras 30(b), 31(a), e 31(b) destacam as respectivas curvas, polinômios e D^2 ajustados. As Tabelas 22, 23 e 24 exibem os resultados dos modelos. O polinômio que expressa o comportamento do algoritmo no Modelo 4 é: $maxCost(x) = 8 - 6x^2 - 1x^4$, onde x é $percNeg$. Já os Modelos 5 e 6 têm o mesmo polinômio: $maxCost(x) = 7 - 4x^2 + 1x^3 - 1x^4$, onde x é $percNeg$.

Tabela 22 – RSFB $maxCost(percNeg)$ para *computational cost* - Amostra 1.

Modelo 4 (M4)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	7.9514	0.1194	66.62	0.0000
poly(x, 4)1	-0.6855	0.4000	-1.71	0.0866
poly(x, 4)2	-6.1049	0.4008	-15.23	0.0000
poly(x, 4)3	0.7411	0.3992	1.86	0.0634
poly(x, 4)4	-1.1437	0.3970	-2.88	0.0040

Tabela 23 – SAAFB $maxCost(percNeg)$ para *computational cost* - Amostra 1.

Modelo 5 (M5)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	7.0701	0.0799	88.45	0.0000
poly(x, 4)1	-0.5058	0.2691	-1.88	0.0602
poly(x, 4)2	-3.7575	0.2696	-13.94	0.0000
poly(x, 4)3	0.9398	0.2681	3.51	0.0005
poly(x, 4)4	-0.8128	0.2662	-3.05	0.0023

Tabela 24 – HS $maxCost(percNeg)$ para *computational cost* - Amostra 1.

Modelo 6 (M6)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	7.0826	0.0796	88.94	0.0000
poly(x, 4)1	-0.4734	0.2681	-1.77	0.0774
poly(x, 4)2	-3.7338	0.2686	-13.90	0.0000
poly(x, 4)3	0.9309	0.2671	3.49	0.0005
poly(x, 4)4	-0.8362	0.2652	-3.15	0.0016

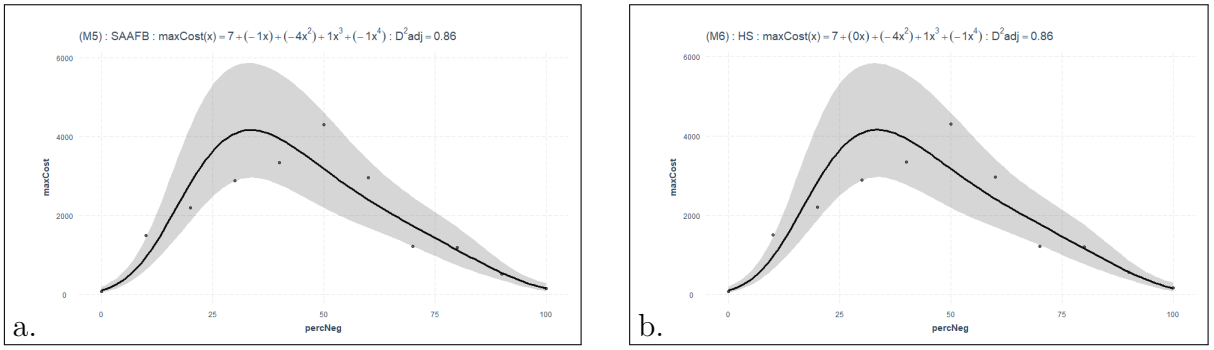

Figura 31 – SAAFB e HS $maxCost(percNeg)$ para *computational cost* - Amostra 1.

Além dos modelos de um fator, três modelos de dois fatores foram desenvolvidos para *computational cost*. As Figuras 32, 33 e 34 destacam cada um desses modelos com seus dados originais. Os eixos dos modelos se referem a $x(vertices)$, $z(percNeg)$ e $y(maxCost)$. A Tabela 25 se refere ao primeiro modelo com dois fatores, ou seja, $maxCost(vertices, percNeg)$ para RSFB. No Modelo 15 (M15), o polinômio que expressa o comportamento do algoritmo é: $maxCost(x, y) = 6+19x-3y-3x^2-35y^2+4y^3-3y^4-4y^6$, onde x é $vertices$ e y é $percNeg$. Já o D^2 ajustado de M15 foi de 88%.

Tabela 25 – RSFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

Modelo 15 (M15)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	6.3092	0.0184	343.15	0.0000
poly(x, 2)1	18.9898	0.4930	38.52	0.0000
poly(x, 2)2	-3.0871	0.4924	-6.27	0.0000
poly(y, 6)1	-3.1256	0.4989	-6.27	0.0000
poly(y, 6)2	-35.3618	0.4999	-70.74	0.0000
poly(y, 6)3	3.5188	0.4972	7.08	0.0000
poly(y, 6)4	-3.3180	0.4937	-6.72	0.0000
poly(y, 6)5	-0.4624	0.4906	-0.94	0.3459
poly(y, 6)6	-3.6113	0.4887	-7.39	0.0000

As Tabelas 26 e 27 se referem aos Modelos 16 (M16) e 17 (M17). Esses modelos têm o mesmo polinômio: $maxCost(x) = 6 + 16x - 3x^2 - 22y^2 + 6y^3 - 3y^4 + 2y^5 + 2y^6$,

onde x se refere ao parâmetro *vertices* e y ao parâmetro $percNeg$. Além disso, M16 e M17 têm D^2 ajustado no valor de 89%.

Tabela 26 – HS $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

Modelo 16 (M16)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	5.7726	0.0127	454.17	0.0000
poly(x, 2)1	16.5659	0.3412	48.55	0.0000
poly(x, 2)2	-2.7747	0.3405	-8.15	0.0000
poly(y, 6)1	-0.6380	0.3469	-1.84	0.0659
poly(y, 6)2	-22.4096	0.3474	-64.51	0.0000
poly(y, 6)3	5.8386	0.3451	16.92	0.0000
poly(y, 6)4	-3.2432	0.3418	-9.49	0.0000
poly(y, 6)5	1.8596	0.3390	5.49	0.0000
poly(y, 6)6	-2.5607	0.3372	-7.59	0.0000

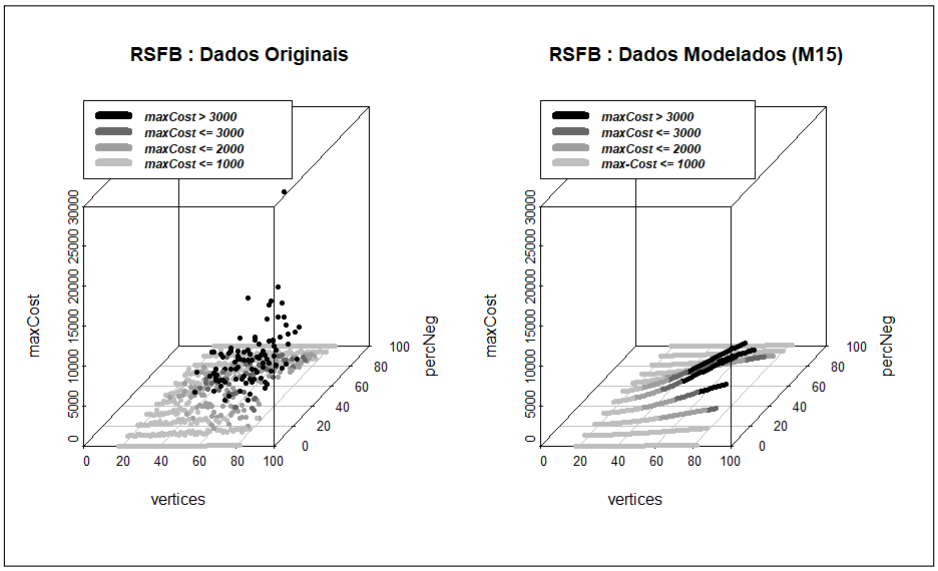

Figura 32 – RSFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

Tabela 27 – SAAFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

Modelo 17 (M17)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	5.7726	0.0127	454.17	0.0000
poly(x, 2)1	16.5659	0.3412	48.55	0.0000
poly(x, 2)2	-2.7747	0.3405	-8.15	0.0000
poly(y, 6)1	-0.6380	0.3469	-1.84	0.0659
poly(y, 6)2	-22.4096	0.3474	-64.51	0.0000
poly(y, 6)3	5.8386	0.3451	16.92	0.0000
poly(y, 6)4	-3.2432	0.3418	-9.49	0.0000
poly(y, 6)5	1.8596	0.3390	5.49	0.0000
poly(y, 6)6	-2.5607	0.3372	-7.59	0.0000

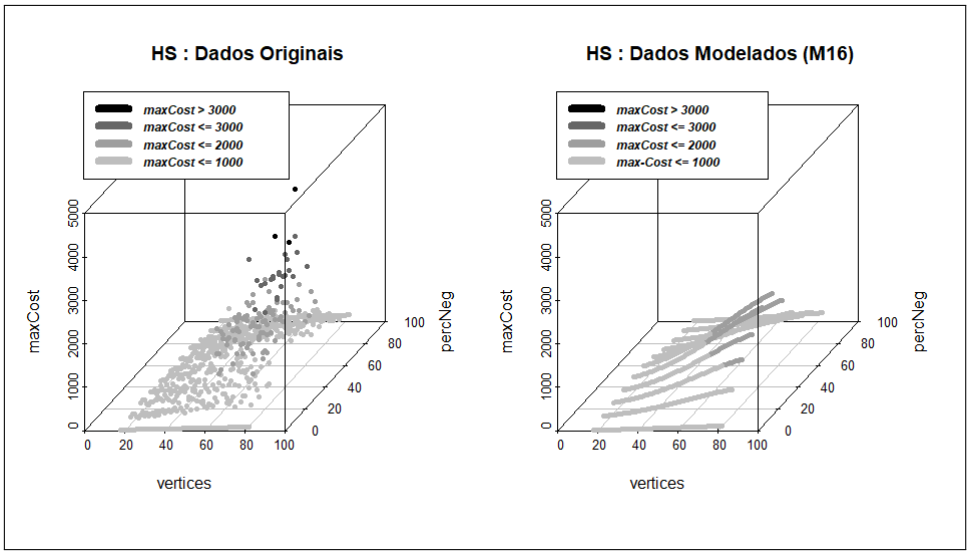

Figura 33 – HS $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

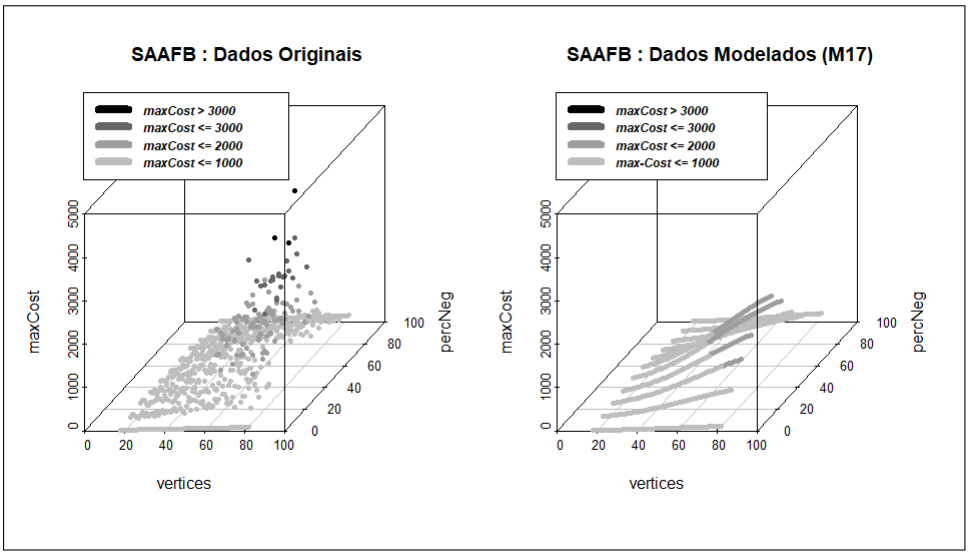

Figura 34 – SAAFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 1.

Neste ponto, cabe dizer que modelos com o fator *edges* também foram testados para RSFB, SAAFB e HS. No entanto, tais resultados não seguem detalhados pois foram muito semelhantes aos dos modelos com o fator *vertices*. Atribui-se à essa semelhança a proporcionalidade que *vertices* e *edges* possuem entre si nas redes da Amostra 1. Consequentemente, na Amostra 1, modelos com o fator *edges* também resultam em polinômios de grau 3 para RSFB e grau 2 para SAAFB e HS.

6.7.1.2 Modelos para *restarted search* (Amostra 1)

Para a métrica *restarted search*, na Amostra 1, foram construídos três modelos. Os Modelos 1b (M1b), 2b (M2b) e 3b (M3b) se referem à função $maxCost(vertices)$ dos algoritmos RSFB, SAAFB e HS, conforme mostram as Figuras 35 e 36. A Tabela 28 mostra que o polinômio do Modelo 1b é: $maxCost(x) = 3 + 0x$, onde x é $vertices$. As Tabelas 29 e 30 mostram que os Modelos 2b e 3b têm o mesmo polinômio: $maxCost(x) = 1 + 0x$, onde x é $vertices$.

Tabela 28 – RSFB $maxCost(vertices)$ para *restarted search* - Amostra 1.

Modelo 1b (M1b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	3.0443	0.1194	25.49	0.0000
x	0.0278	0.0022	12.67	0.0000

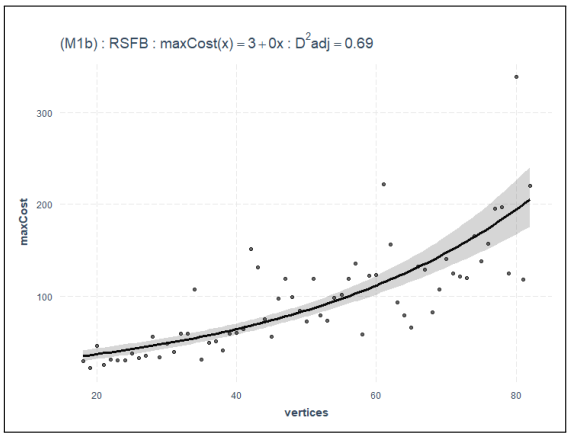

Figura 35 – RSFB $maxCost(vertices)$ para *restarted search* - Amostra 1.

Tabela 29 – SAAFB $maxCost(vertices)$ para *restarted search* - Amostra 1.

Modelo 2b (M2b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	1.5110	0.1349	11.20	0.0000
x	0.0109	0.0024	4.59	0.0000

Tabela 30 – HS $maxCost(vertices)$ para *restarted search* - Amostra 1.

Modelo 3b (M3b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	1.4908	0.1355	11.00	0.0000
x	0.0112	0.0024	4.70	0.0000

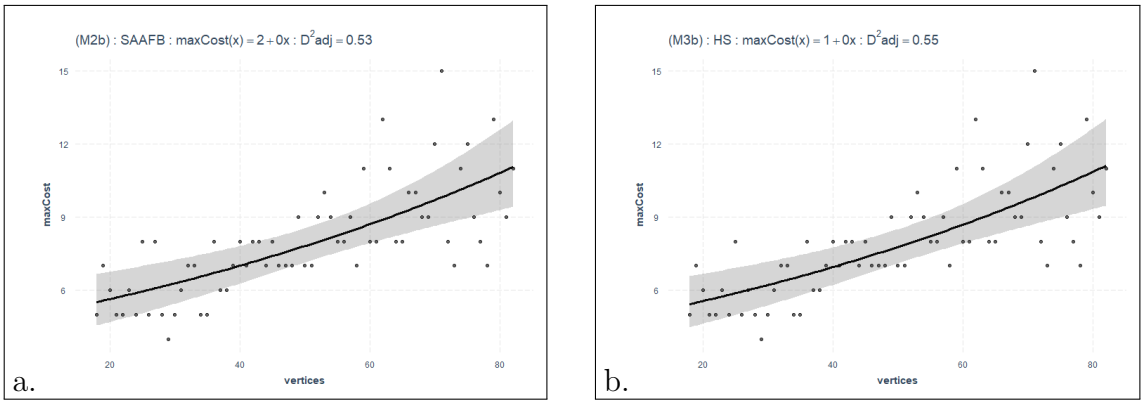

Figura 36 – SAAFB e HS $maxCost(vertices)$ para *restarted search* - Amostra 1.

6.7.2 Modelos da Amostra 2

Os resultados da Amostra 2 (redes entre 16 e 320 vértices) permitiram a criação de oito modelos: seis referentes à métrica *computational cost* e dois referentes à métrica *restarted search*.

6.7.2.1 Modelos para *computational cost* (Amostra 2)

Os Modelos 7 (M7) e 8 (M8) se referem à função $maxCost(vertices)$ dos algoritmos SAAFB e HS para a métrica *computational cost*, conforme mostra a Figura 37. As Tabelas 31 e 32 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 9+22x-6x^2+2x^3$, onde x é *vertices*.

Tabela 31 – SAAFB $maxCost(vertices)$ para *computational cost* - Amostra 2.

Modelo 7 (M7)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	8.7534	0.0228	383.93	0.0000
poly(x, 3)1	22.4161	0.3987	56.23	0.0000
poly(x, 3)2	-6.0673	0.3987	-15.22	0.0000
poly(x, 3)3	2.2616	0.3987	5.67	0.0000

Tabela 32 – HS $maxCost(vertices)$ para *computational cost* - Amostra 2.

Modelo 8 (M8)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	8.7560	0.0228	384.63	0.0000
poly(x, 3)1	22.3701	0.3981	56.20	0.0000
poly(x, 3)2	-6.0328	0.3981	-15.15	0.0000
poly(x, 3)3	2.2387	0.3981	5.62	0.0000

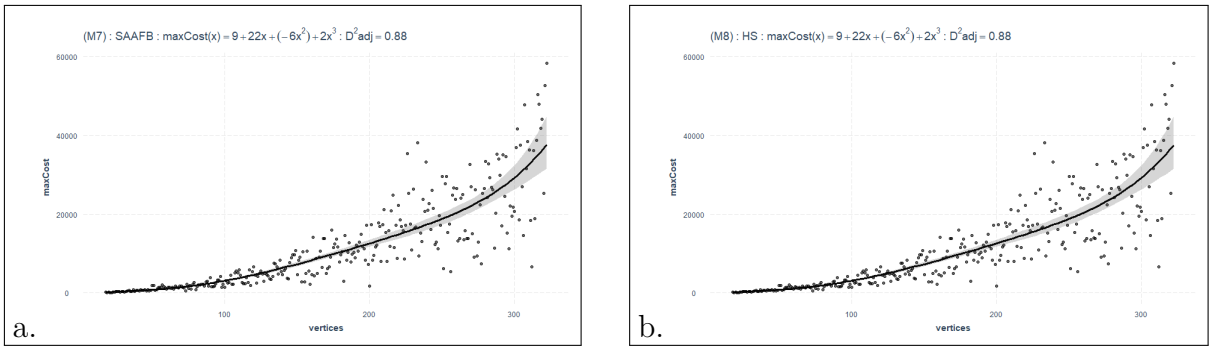

Figura 37 – SAAFB, HS $maxCost(vertices)$ para *computational cost* - Amostra 2.

Os Modelos 9 (M9) e 10 (M10) se referem ao $maxCost(percNeg)$ dos algoritmos SAAFB e HS, conforme mostra a Figura 38. As Tabelas 33 e 34 destacam que M9 e M10 têm o mesmo polinômio: $maxCost(x) = 9 - 1x - 5x^2 + x^3 - 1x^4 + 1x^5 - 1x^6$, onde x é $percNeg$.

Tabela 33 – SAAFB $maxCost(percNeg)$ para *computational cost* - Amostra 2.

Modelo 9 (M9)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	9.1172	0.0349	260.94	0.0000
poly(x, 6)1	-0.8530	0.1187	-7.19	0.0000
poly(x, 6)2	-5.1860	0.1191	-43.54	0.0000
poly(x, 6)3	1.3668	0.1181	11.57	0.0000
poly(x, 6)4	-0.6875	0.1166	-5.89	0.0000
poly(x, 6)5	0.5134	0.1154	4.45	0.0000
poly(x, 6)6	-0.7549	0.1146	-6.58	0.0000

Tabela 34 – HS $maxCost(percNeg)$ para *computational cost* - Amostra 2.

Modelo 10 (M10)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	9.1186	0.0349	261.04	0.0000
poly(x, 6)1	-0.8529	0.1186	-7.19	0.0000
poly(x, 6)2	-5.1829	0.1191	-43.53	0.0000
poly(x, 6)3	1.3665	0.1181	11.57	0.0000
poly(x, 6)4	-0.6871	0.1166	-5.89	0.0000
poly(x, 6)5	0.5138	0.1154	4.45	0.0000
poly(x, 6)6	-0.7546	0.1146	-6.58	0.0000

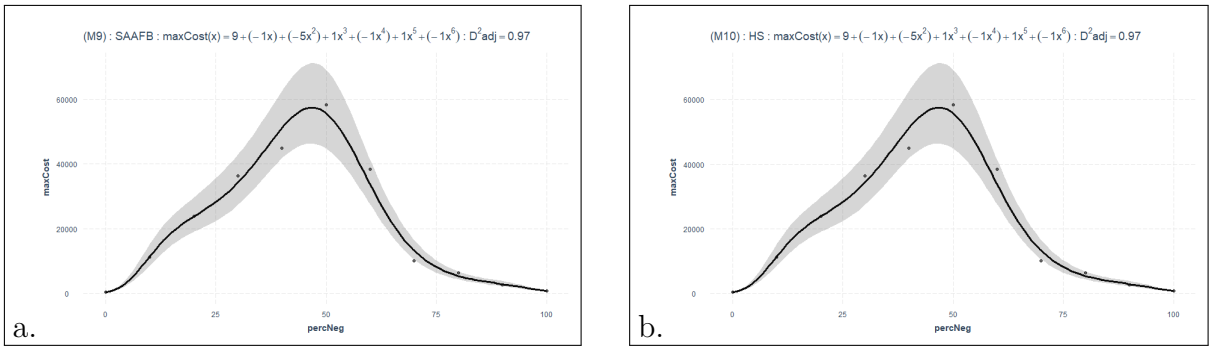

Figura 38 – SAAFB, HS $maxCost(percNeg)$ para *computational cost* - Amostra 2.

Com dois fatores, os Modelos 18 (M18) e 19 (M19) se referem à função $maxCost(vertices, percNeg)$ dos algoritmos HS e SAAFB, conforme mostram as Figuras 39 e 40. As Tabelas 35 e 36 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 7 + 58x - 6y - 15x^2 - 61y^2 + 5x^3 + 19y^3 - 1y^4 + 1y^5 - 6y^6$, onde x é $vertices$ e y é $percNeg$. M18 e M19 apresentam o mesmo valor para D^2 ajustado, 42%.

Tabela 35 – HS $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 2.

Modelo 18 (M18)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	6.9367	0.0262	265.07	0.0000
poly(x, 3)1	57.9835	1.5165	38.24	0.0000
poly(x, 3)2	-14.8337	1.5165	-9.78	0.0000
poly(x, 3)3	5.1964	1.5164	3.43	0.0006
poly(y, 6)1	-5.7769	1.5168	-3.81	0.0001
poly(y, 6)2	-60.6295	1.5169	-39.97	0.0000
poly(y, 6)3	18.8305	1.5165	12.42	0.0000
poly(y, 6)4	-0.8386	1.5161	-0.55	0.5802
poly(y, 6)5	1.0439	1.5157	0.69	0.4910
poly(y, 6)6	-6.1972	1.5154	-4.09	0.0000

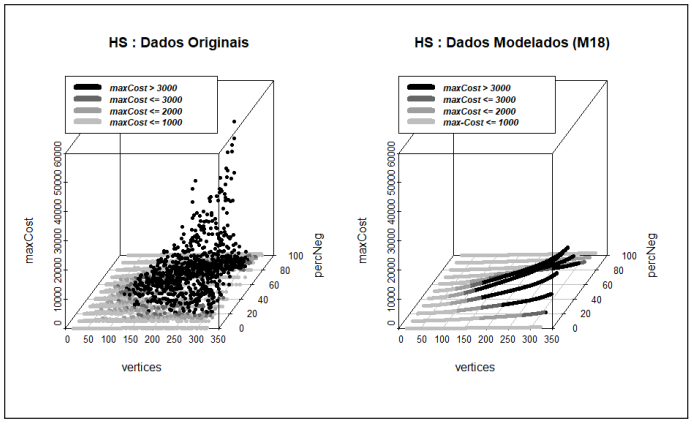

Figura 39 – HS $maxCost(vertices, percNeg)$ para *comp. cost* - Amostra 2.

Tabela 36 – SAAFB $maxCost(vertices, percNeg)$ para *comp. cost* - Amostra 2.

Modelo 19 (M19)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	6.9308	0.0262	265.00	0.0000
poly(x, 3)1	58.2700	1.5156	38.45	0.0000
poly(x, 3)2	-15.0443	1.5156	-9.93	0.0000
poly(x, 3)3	5.3331	1.5156	3.52	0.0004
poly(y, 6)1	-5.7890	1.5160	-3.82	0.0001
poly(y, 6)2	-60.7725	1.5160	-40.09	0.0000
poly(y, 6)3	18.8451	1.5157	12.43	0.0000
poly(y, 6)4	-0.8444	1.5152	-0.56	0.5774
poly(y, 6)5	1.0551	1.5148	0.70	0.4861
poly(y, 6)6	-6.1917	1.5145	-4.09	0.0000

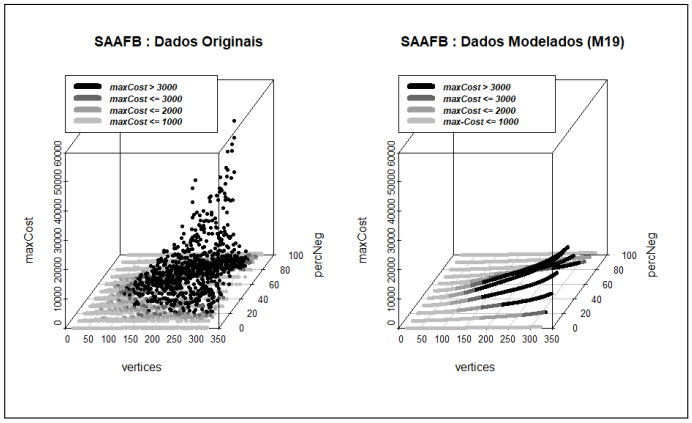

Figura 40 – SAAFB $maxCost(vertices, percNeg)$ para *comp. cost* - Amostra 2.

Modelos para o fator *edges* também foram testados para SAAFB e HS. No entanto, assim como na Amostra 1, seus resultados foram semelhantes aos modelos com o fator *vertices* devido à proporcionalidade entre *vertices* e *edges*. Em outras palavras, na Amostra 2, os modelos com o fator *edges* resultaram em polinômios com grau 3 para SAAFB e HS.

6.7.2.2 Modelos para *restarted search* (Amostra 2)

Para a métrica *restarted search*, dois modelos foram construídos na Amostra 2. Os Modelos 7b (M7b) e 8b (M8b) se referem a $maxCost(vertices)$ de SAAFB e HS, conforme indica a Figura 41. As Tabelas 37 e 38 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 2 + 0x$, onde x é *vertices* .

Tabela 37 – SAAFB $maxCost(vertices)$ para *restarted search* - Amostra 2.

Modelo 7b (M7b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	1.7439	0.0457	38.16	0.0000
x	0.0034	0.0002	15.81	0.0000

Tabela 38 – HS $maxCost(vertices)$ para *restarted search* - Amostra 2.

Modelo 8b (M8b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	1.7439	0.0457	38.16	0.0000
x	0.0034	0.0002	15.81	0.0000

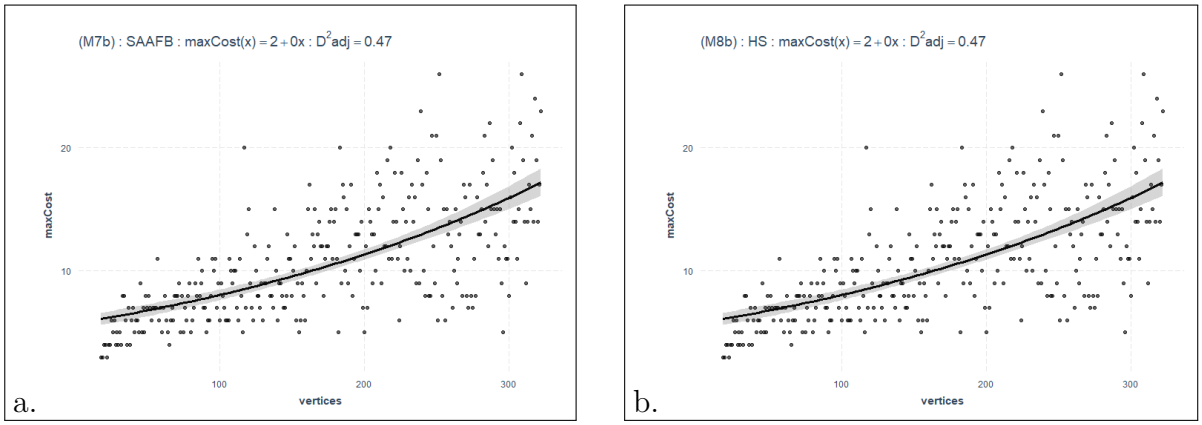

Figura 41 – SAAFB e HS $maxCost(vertices)$ para *restarted search* - Amostra 2.

6.7.3 Modelos da Amostra 3

Na Amostra 3 (redes entre 16 e 320 vértices, mas como dígrafos bipartidos completos), foram elaborados oito modelos: seis referentes à métrica *computational cost* e dois referentes à métrica *restarted search*.

6.7.3.1 Modelos para *computational cost* (Amostra 3)

Os Modelos 11 (M11) e 12 (M12) se referem à função $maxCost(vertices)$ dos algoritmos SAAFB e HS, conforme indica a Figura 46. As Tabelas 39 e 40 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 13 + 31x - 11x^2 + 6x^3$, onde x é $vertices$.

Tabela 39 – SAAFB $maxCost(vertices)$ para *computational cost* - Amostra 3.

Modelo 11 (M11)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	13.2331	0.0293	451.06	0.0000
poly(x, 3)1	31.5093	0.5124	61.50	0.0000
poly(x, 3)2	-10.8626	0.5124	-21.20	0.0000
poly(x, 3)3	5.7247	0.5124	11.17	0.0000

Tabela 40 – HS $maxCost(vertices)$ para *computational cost* - Amostra 3.

Modelo 12 (M12)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	13.2335	0.0293	451.34	0.0000
poly(x, 3)1	31.4995	0.5121	61.51	0.0000
poly(x, 3)2	-10.8533	0.5121	-21.19	0.0000
poly(x, 3)3	5.7168	0.5121	11.16	0.0000

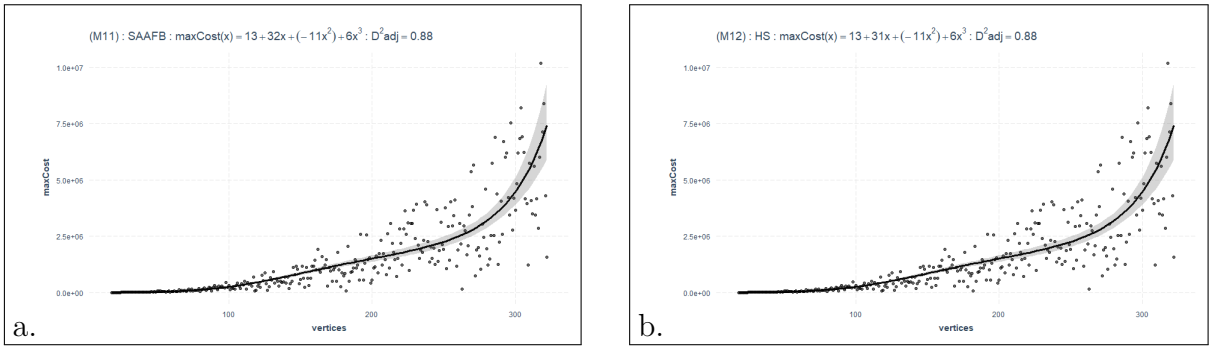

Figura 42 – SAAFB e HS $maxCost(vertices)$ para *computational* - Amostra 3.

Os Modelos 13 (M13) e 14 (M14) se referem à função $maxCost(percNeg)$ de SAAFB e HS, conforme indica a Figura 43. As Tabelas 41 e 42 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 13 + 9 - 4^2 + 2x^3 - 1x^4 + 1x^5 - 1x^6$, onde x é $percNeg$.

Tabela 41 – SAAFB $maxCost(percNeg)$ para *computational cost* - Amostra 3.

Modelo 13 (M13)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	13.3799	0.0341	392.41	0.0000
poly(x, 6)1	8.7585	0.1149	76.22	0.0000
poly(x, 6)2	-4.0915	0.1153	-35.47	0.0000
poly(x, 6)3	1.7654	0.1147	15.40	0.0000
poly(x, 6)4	-0.6835	0.1136	-6.02	0.0000
poly(x, 6)5	0.7251	0.1127	6.44	0.0000
poly(x, 6)6	-0.6181	0.1121	-5.51	0.0000

Tabela 42 – HS $maxCost(percNeg)$ para *computational cost* - Amostra 3.

Modelo 14 (M14)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	13.3803	0.0341	392.54	0.0000
poly(x, 6)1	8.7567	0.1149	76.23	0.0000
poly(x, 6)2	-4.0899	0.1153	-35.47	0.0000
poly(x, 6)3	1.7642	0.1146	15.39	0.0000
poly(x, 6)4	-0.6827	0.1135	-6.01	0.0000
poly(x, 6)5	0.7246	0.1126	6.43	0.0000
poly(x, 6)6	-0.6179	0.1121	-5.51	0.0000

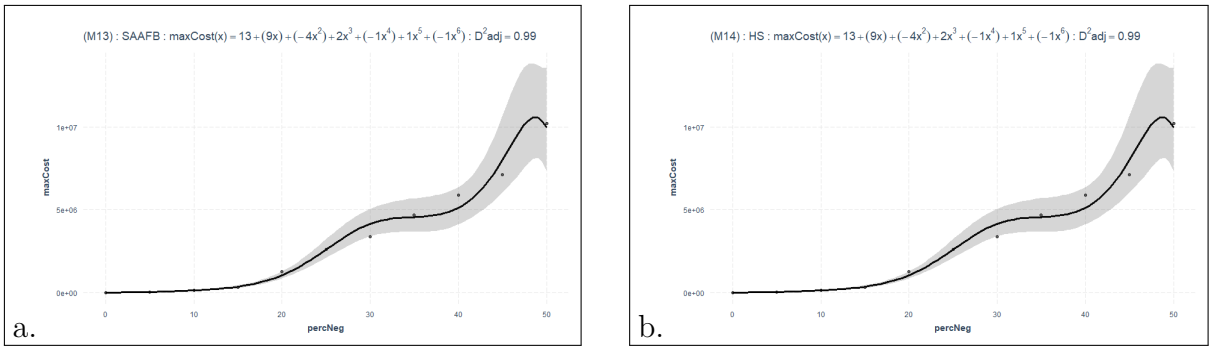

Figura 43 – SAAFB e HS $maxCost(percNeg)$ para *computational cost* - Amostra 3.

Os Modelos 20 (M20) e 21 (M21) se referem à função $maxCost(vertices, percNeg)$ de HS e SAAFB, na Amostra 3, conforme indicam as Figuras 44 e 45. As Tabelas 43 e 44 demonstram que os modelos têm o mesmo polinômio: $maxCost(x, y) = 10 + 86x + 120y - 26x^2 - 48y^2 + 12x^3 + 17y^3 - 7y^4 + 5y^5$, onde x é $vertices$ e y é $percNeg$.

Tabela 43 – HS $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 3.

Modelo 20 (M20)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	10.5218	0.0382	275.52	0.0000
poly(x, 3)1	85.6790	2.2126	38.72	0.0000
poly(x, 3)2	-25.1301	2.2127	-11.36	0.0000
poly(x, 3)3	11.9334	2.2127	5.39	0.0000
poly(y, 5)1	119.9360	2.2127	54.20	0.0000
poly(y, 5)2	-47.6153	2.2129	-21.52	0.0000
poly(y, 5)3	17.3401	2.2126	7.84	0.0000
poly(y, 5)4	-7.3708	2.2122	-3.33	0.0009
poly(y, 5)5	5.3848	2.2119	2.43	0.0149

Tabela 44 – SAAFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 3.

Modelo 21 (M21)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	10.5176	0.0382	275.03	0.0000
poly(x, 3)1	85.8604	2.2156	38.75	0.0000
poly(x, 3)2	-25.3113	2.2158	-11.42	0.0000
poly(x, 3)3	12.0898	2.2157	5.46	0.0000
poly(y, 5)1	119.9602	2.2158	54.14	0.0000
poly(y, 5)2	-47.8649	2.2159	-21.60	0.0000
poly(y, 5)3	17.3284	2.2156	7.82	0.0000
poly(y, 5)4	-7.5178	2.2152	-3.39	0.0007
poly(y, 5)5	5.3595	2.2149	2.42	0.0155

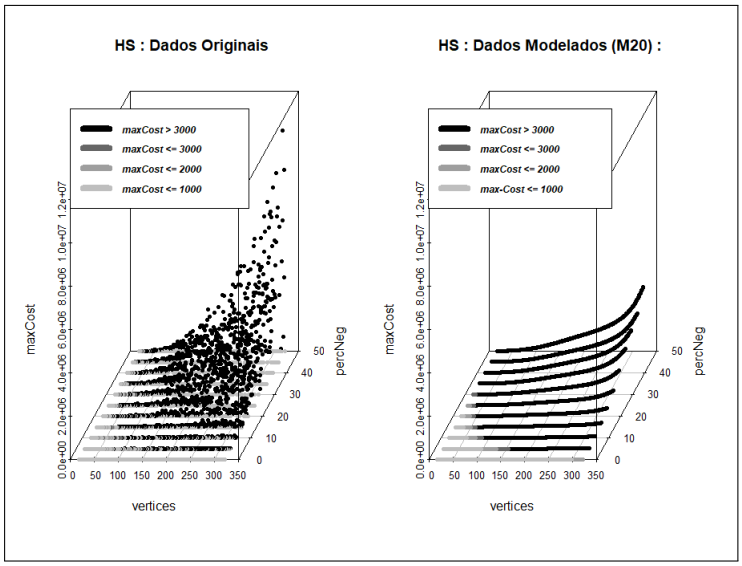

Figura 44 – HS $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 3.

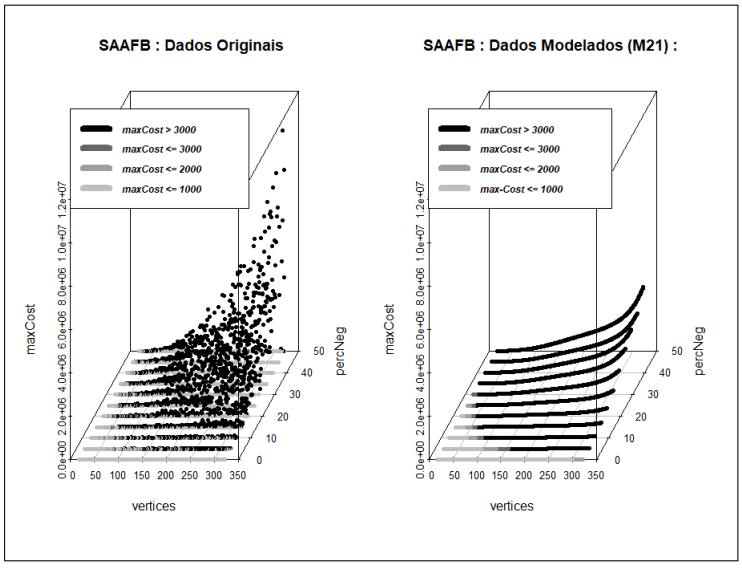

Figura 45 – SAAFB $maxCost(vertices, percNeg)$ para *computational cost* - Amostra 3.

Na Amostra 3, modelos para o fator *edges* também foram testados. No entanto, seus resultados mostraram baixo poder explicativo com D^2 ajustado.

6.7.3.2 Modelos para *restarted search* (Amostra 3)

Para a métrica *restarted search* foram construídos mais dois modelos. Os Modelos 11b (M11b) e 12b (M12b) se referem à função $maxCost(vertices)$ de SAAFB e HS, conforme indica a Figura 46. As Tabelas 45 e 46 mostram que os modelos têm o mesmo polinômio: $maxCost(x) = 3 + 0x$, onde x é $vertices$.

Tabela 45 – SAAFB $maxCost(vertices)$ para *restarted search* - Amostra 3.

Modelo 11b (M11b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	3.3878	0.0402	84.18	0.0000
x	0.0055	0.0002	26.74	0.0000

Tabela 46 – HS $maxCost(vertices)$ para *restarted search* - Amostra 3.

Modelo 12b (M12b)	Estimativa	Erro padrão	valor z	Pr(>|z|)
(Intercepto)	3.3829	0.0404	83.73	0.0000
x	0.0055	0.0002	26.74	0.0000

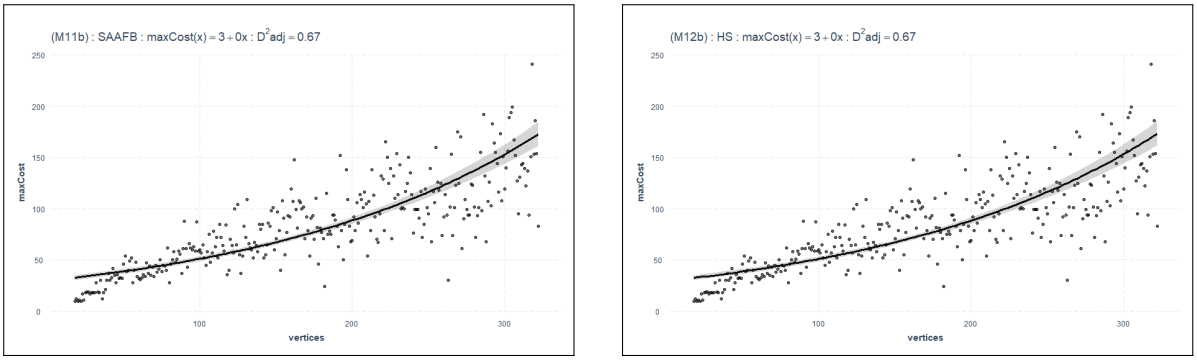

Figura 46 – SAAFB e HS $maxCost(vertices)$ para *restarted search* - Amostra 3.

7 ANÁLISE E DISCUSSÃO

Neste capítulo os resultados do experimento são analisados e discutidos. Para isso, os resultados dos modelos foram resumidos por amostra. Então, os resultados seguem organizados por métrica, fatores da função *maxCost* (*vertices*; *percNeg*; e *vertices* com *percNeg*) e algoritmos (RSFB, SAAFB e HS). Por questão de ênfase, os resultados dos modelos seguem reduzidos aos termos de maior grau dos respectivos polinômios, destacando a ordem de grandeza em cada caso.

7.1 DISCUSSÃO SOBRE A AMOSTRA 1

Em primeiro lugar, importa lembrar que a métrica *computational cost* se refere ao custo total de cada algoritmo e a métrica *restarted search* se refere ao número de vezes que a busca é reiniciada na árvore geradora para cada algoritmo. Sobre a Amostra 1, a Tabela 47 resume todos os modelos elaborados, onde as três primeiras linhas destacam os modelos para a métrica *computational cost* (como *comp. cost*) e a última linha mostra os modelos para a métrica *restarted search* (como *rest. search*). Assim, na primeira linha, para *computational cost* com $maxCost(vertices)$ o desempenho de RSFB foi da ordem de $vertices^3$ e os desempenhos de SAAFB e HS foram da ordem de $vertices^2$ (destacados com sublinhado). Na segunda linha, para *computational cost* com $maxCost(percNeg)$ todos os algoritmos mostraram o mesmo desempenho, ou seja, $percNeg^4$. Na terceira linha, para *computational cost* com $maxCost(vertices, percNeg)$ todos os resultados foram $vertices^2 + percNeg^6$. Na última linha, para a *restarted search* com $maxCost(vertices)$ todos os algoritmos mostraram os mesmos resultados, ou seja, ordem de $vertices^1$.

Cabe resgatar que o teste de KS (descrito na Tabela 10) mostrou na Amostra 1 para a métrica *computational cost* RSFB $\not\sim$ SAAFB, RSFB $\not\sim$ SAAFB e HS $\sim$ SAAFB. Isso apoia os resultados da Tabela 47. Em outras palavras, o grau do polinômio de RSFB é diferente dos graus dos polinômios de SAAFB e HS, quando a métrica é *computational cost*, assim como RSFB tem distribuição estatisticamente diferente das distribuições de SAAFB e HS. Ao mesmo tempo, SAAFB e HS têm o mesmo grau de polinômio e distribuições estatisticamente semelhantes.

Tabela 47 – Maior grau dos polinômios por métrica - Amostra 1.

Métrica	*maxCost* Fator	RSFB grau	SAAFB grau	HS grau
comp. cost	*vertices*	$vertices^3$	$\underline{vertices^2}$	$\underline{vertices^2}$
comp. cost	*percNeg*	$percNeg^4$	$percNeg^4$	$percNeg^4$
comp. cost	*vertices, percNeg*	$vert.^2 + percNeg^6$	$vert.^2 + percNeg^6$	$vert.^2 + percNeg^6$
rest. search	*vertices*	$vertices^1$	$vertices^1$	$vertices^1$

7.2 DISCUSSÃO SOBRE A AMOSTRA 2

Semelhante à seção anterior, os resultados dos modelos na Amostra 2 estão resumidos na Tabela 48. Na primeira linha dessa tabela, para *computational cost* com $maxCost(vertices)$, os resultados de SAAFB e HS foram da ordem de $vertices^3$. Na segunda linha, para *computational cost* com $maxCost(percNeg)$ os resultados de SAAFB e HS foram $percNeg^6$. Na terceira linha, para a métrica *computational cost* com a função $maxCost(vertices, percNeg)$ os resultados foram da ordem de $vertices^2 + percNeg^6$. Na última linha, para *restarted search* com $maxCost(vertices)$, os resultados de SAAFB e HS foram da ordem de $vertices^1$, como na Amostra 1.

Em relação ao teste de KS, de acordo com a Tabela 10 (na Amostra 2) para *computational cost*, SAAFB $\sim$ HS. Esse resultado está em harmonia com os resultados da Tabela 48. Em outras palavras, os graus máximos dos polinômios são os mesmos em todas as funções $maxCost()$ entre SAAFB e HS.

Tabela 48 – Maior grau dos polinômios por métrica - Amostra 2.

Métrica	*maxCost* Fator	SAAFB grau	HS grau
computational cost	*vertices*	$vertices^3$	$vertices^3$
computational cost	*percNeg*	$percNeg^6$	$percNeg^6$
computational cost	$vertices, percNeg$	$vertices^3 + percNeg^6$	$vertices^3 + percNeg^6$
restarted search	*vertices*	$vertices^1$	$vertices^1$

7.3 DISCUSSÃO SOBRE A AMOSTRA 3

Na Amostra 3, para *computational cost* com $maxCost(vertices)$, através da Tabela 49, pode ser observado que os polinômios de SAAFB e HS não ultrapassam o grau 3. Na terceira linha, para *computational cost* com $maxCost(vertices, percNeg)$, os resultados foram $vertices^2 + percNeg^5$. Os outros resultados foram os mesmos da Amostra 2.

Tabela 49 – Maior grau dos polinômios por métrica - Amostra 3.

Métrica	*maxCost* Fator	SAAFB grau	HS grau
computational cost	*vertices*	$vertices^3$	$vertices^3$
computational cost	*percNeg*	$percNeg^6$	$percNeg^6$
computational cost	*vertices, percNeg*	$vertices^3 + percNeg^5$	$vertices^3 + percNeg^5$
restarted search	*vertices*	$vertices^1$	$vertices^1$

8 CONCLUSÕES

Este capítulo apresenta a síntese da realização da pesquisa, a resposta geral dada à situação problema, recobra as hipóteses nulas e alternativas, aponta as principais conclusões e suas relações com as hipóteses, bem como apresenta sugestões para estudos futuros.

8.1 SÍNTESE DA PESQUISA

Neste trabalho, três algoritmos (RSFB, SAAFB e HS) para o problema de escalonamento $^{\circ} \mid cpm, \delta_n, c_j \mid max\text{-}npv$, descrito no Capítulo 3, foram submetidos a um experimento fatorial. A estrutura do experimento empregou uma função chamada $maxCost$, cujos os parâmetros eram fatores de rede como $vertices$ e $percNeg$ (porcentagem de atividades negativas). Os resultados foram expressos por meio de duas métricas: 1) *computational cost* e 2) *restarted search*. Enquanto a primeira métrica se referia ao custo total de cada um dos algoritmos, a segunda métrica se referia ao número de reinicializações de buscas na árvore geradora para cada algoritmo.

Os resultados foram obtidos usando três amostras de diferentes configurações de rede. Na primeira amostra foram utilizados grafos com 16 a 80 vértices, randomizados em termos de vértices, camadas, porcentagem de atividades negativas, entre outros parâmetros. A segunda amostra utilizou grafos com 16 a 320 vértices e randomização semelhante à primeira amostra. Por fim, a terceira amostra se referiu a um conjunto de grafos de conveniência para estressar os algoritmos com redes contidas com alto número de arestas. Nesse caso, foram utilizados dígrafos bipartidos completos (desconsiderando *dummies*) com vértices entre 16 e 320.

8.2 RESPOSTA GERAL À SITUAÇÃO PROBLEMA

No primeiro capítulo desta tese, a situação problema foi apresentada através das seguintes perguntas:

- **É possível realizar um experimento que evidencie a ordem de complexidade global (Notação O) dos algoritmos, considerando a incorporação da estratégia de escolha da direção da busca nos três algoritmos (RS, SAA e HS)?**

- **É possível realizar um experimento que evidencie a ordem de complexidade do número de buscas nas respectivas árvores geradoras, considerando a incorporação da estratégia de escolha da direção da busca nos três algoritmos (RS, SAA e HS)?**

Quanto à primeira pergunta da situação problema, através dos modelos estatísticos globais do experimento, foi possível constatar a ordem de complexidade global de cada um dos três algoritmos, considerando os dados das três amostras utilizadas.

Quanto à segunda pergunta da situação problema, com apoio dos modelos estatísticos referentes ao número de buscas nas respectivas árvores geradoras, também foi possível constatar a ordem de complexidade dessas buscas em cada um dos três algoritmos, considerando os dados das três amostras.

8.3 HIPÓTESES NULAS *VS* ALTERNATIVAS

Antes de anunciar as principais conclusões, é válido relembrar os textos de cada uma das hipóteses (nulas H_0 e alternativas H_a), para facilitar a indicação das relações entre conclusões e hipóteses.

- **Hipótese I_0 ($H1_0$):**
  O algoritmo RSFB **não está** em ordem (classe) de complexidade global maior que os algoritmos SAAFB e HS.
- **Hipótese I_a ($H1_a$):**
  O algoritmo RSFB **está** em ordem (classe) de complexidade global maior que os algoritmos SAAFB e HS.
- **Hipótese II_0 ($H2_0$):**
  Os algoritmos HS e SAAFB **não estão** na mesma ordem (classe) de complexidade global.
- **Hipótese II_a ($H2_a$):**
  Os algoritmos HS e SAA **estão** na mesma ordem (classe) de complexidade global.
- **Hipótese III_0 ($H3_0$):**
  Os algoritmos RSFB, SAAFB e HS **não estão** na mesma ordem (classe) de complexidade, no que se refere ao número de buscas nas respectivas árvores geradoras.
- **Hipótese III_a ($H3_a$):**
  Os algoritmos RSFB, SAAFB e HS **estão** na mesma ordem (classe) de complexidade, no que se refere ao número de buscas nas respectivas árvores geradoras.

8.4 CONCLUSÕES E RELAÇÕES COM AS HIPÓTESES

8.4.1 SAAFB e HS superam RSFB na primeira métrica

Conforme indicado, o algoritmo RSFB foi utilizado apenas nas redes da Amostra 1. Nesse caso, o maior grau do polinômio identificado para a métrica *computational cost* com $maxCost(vertices)$ foi $vertices^3$, enquanto os algoritmos SAAFB e HS foram expressos em polinômios de grau dois, ou seja, $vertices^2$. Os resultados empíricos mostram que usando $maxCost(vertices)$ como apontador (*proxy*) para $O(vertices)$ podemos estimar que o algoritmo RSFB tem um custo de tempo de $O(vertices^3)$, enquanto SAAFB e HS apresentaram um custo de tempo de $O(vertices^2)$, com as redes da Amostra 1. Além disso, os resultados do teste de KS, conforme Tabela 10, mostraram uma diferença estatisticamente significativa entre as distribuições de RSFB e SAAFB; e RSFB e HS.

Portanto, as constatações de que SAAFB e HS estão na classe de complexidade $O(vertices^2)$ e que RSFB está na classe de complexidade $O(vertices^3)$ apoiam a corroboração da Hipótese I_a (alternativa) e a refutação da Hipótese I_0 (nula). Nos termos da $H1_a$, **RSFB está em ordem (classe) de complexidade global maior que os algoritmos SAAFB e HS**.

8.4.2 SAAFB e HS têm o mesmo desempenho em ambas as métricas

No caso de SAAFB e HS, os resultados do teste de KS (nas três amostras e em ambas as métricas) não apresentaram diferenças estatisticamente significativas, como mostram as comparações fatoriais da função *maxCost* (Tabela 10). Além disso, SAAFB e HS apresentaram valores iguais para o grau máximo de polinômios para todos os fatores e métricas analisadas. Assim, podemos concluir, dentro dos limites de precisão estatística, que tanto SAAFB quanto HS estão na mesma ordem (classe) de complexidade.

Portanto, essa constatação apoia a corroboração da Hipótese II_a (alternativa) e a refutação da Hipótese II_0 nula. Nos termos de $H2_a$, **os algoritmos HS e SAA estão na mesma ordem (classe) de complexidade global**.

8.4.3 RSFB, SAAFB e HS têm desempenhos semelhantes na segunda métrica

Os resultados da métrica *restarted search* em todas as amostras por função *maxCost* dos três algoritmos apresentaram a mesma ordem de custo. No caso, o grau mais alto dos polinômios dos modelos elaborados sempre foi $vertices^1$. Em outras palavras, RSFB,

SAAFB e HS apresentaram um custo de $O(vertices^1)$. É um achado importante, pois se refere a uma das questões em aberto nos três algoritmos.

Assim, essa constatação apoia a corroboração da Hipótese III_a (alternativa) e a refutação da Hipótese III_0 nula. Nos termos de $H3_a$, **os algoritmos RSFB, SAAFB e HS estão na mesma ordem (classe) de complexidade, no que se refere ao número de buscas nas respectivas árvores geradoras**.

8.4.4 Fator mais influente de desempenho

Os modelos que incluem o fator *percNeg* apresentaram polinômios com graus superiores aos que não possuem tal fator. No entanto, o fator *percNeg* considera um intervalo com valores de 0% a 100%. Nesse caso, foi possível verificar que a influência de *percNeg* no custo do escalonamento aumenta até o valor ficar em torno de 50%, depois disso a influência diminui. Por outro lado, o fator *vertices* influencia o custo de escalonamento sem limite definido. Em geral, isso indica que enquanto o fator *vertices* puder crescer, o custo do escalonamento também crescerá sem limite. Portanto, o fator *vertices* exclusivamente é mais apropriado para expressar a ordem de crescimento de custo dos algoritmos nas três amostras com as duas métricas. Nesse sentido, onde n é *vertices*, o custo de escalonamento foi: a) $O(n^3)$ para RSFB, com a métrica *computational cost* na Amostra 1; b) $O(n^2)$, com a métrica *computational cost* para SAAFB e HS, na Amostra 1; c) $O(n^3)$, com a métrica *computational cost* para SAAFB e HS, nas Amostras 2 e 3; d) e $O(n)$, com a métrica *restarted search* para RSFB (na Amostra 1), SAAFB e HS (nas três amostras).

8.4.5 Resultados da tese diante das lacunas prévias

Os resultados do experimento desta pesquisa podem ser colocados diante das prévias lacunas da literatura. Nesse sentido, a Tabela 50 destaca o custo global (denominado por *Global*, o que equivale à métrica *computational cost*) e o custo de busca na árvore geradora (denominado por *Árvore ger.*, o que equivale à métrica *restarted search*). No caso, RS_b, SAA_b e HS_b se referem aos algoritmos com suas respectivas lacunas de custo na literatura (problemas considerados abertos). Enquanto isso, $RSFB_a$, $SAAFB_a$ e HS_a se referem aos resultados obtidos a partir do experimento desta tese. Vale lembrar que RS e SAA não têm a estratégia de inversão de busca, mas RSFB e SAAFB incluem essa estratégia.

Tabela 50 – Resultados da tese e as lacunas prévias.

Custo	RS_b	SAA_b	HS_b	$RSFB_a$	$SAAFB_a$	HS_a
Global	aberto	aberto	aberto	$O(n^3)$	$O(n^2)$ ou $O(n^3)$	$O(n^2)$ ou $O(n^3)$
Árvore ger.	aberto	aberto	aberto	$O(n)$	$O(n)$	$O(n)$

Ainda na Tabela 50, o algoritmo RSFB (referido como RSFB_a) apresentou custo global de $O(n^3)$, sendo utilizado exclusivamente nas instâncias da Amostra 1. Os algoritmos SAAFB e HS (referidos como SAAFB_a e HS_a) apresentaram custos globais de $O(n^2)$, para as instâncias da Amostra 1. Para as instâncias das Amostras 2 e 3, SAAFB e HS apresentaram custos globais de $O(n^3)$. Por fim, os custos de busca na árvore geradora, nas três amostras, foram de $O(n)$.

8.4.6 Contribuições publicadas

Alguns dos resultados obtidos durante a realização desta pesquisa foram registrados através das seguintes publicações:

- LACERDA, I. M.; SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R.. Theoretical and empirical analysis of algorithms for the *max-npv* project scheduling problem. In: **10th Latin American Workshop on Cliques of Graphs**, 2022, Curitiba.

- LACERDA, Isac M. SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R. . Empirical Evaluation of Project Scheduling Algorithms for Maximization of the Net Present Value. **ArXiv preprint arXiv:2207.03330**, 2022.

- LACERDA, I. M.; SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R. . Avaliação empírica de algoritmos clássicos para escalonamento de projetos com maximização de valor presente líquido. In: **LIII Simpósio Brasileiro de Pesquisa Operacional (SBPO)**, 2021, João Pessoa.

- LACERDA, I. M.; SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R.. Sobre a Complexidade de um Algoritmo Clássico para Escalonamento de Projetos com Maximização de Valor Presente Líquido. In: **LI Simpósio Brasileiro de Pesquisa Operacional (SBPO)**, 2019, Limeira.

- LACERDA, I. M.; SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R.. Refinamento e Análise da Complexidade Polinomial de um Algoritmo Clássico para o Problema de Escalonamento de Projetos com Maximização do Valor Presente Líquido. In: **Escuela Latinoamericana de Verano de Investigación Operativa (ELAVIO)**, 2019, Llerida, Espanha.

- LACERDA, I. M.; SZWARCFITER, J. L.; SCHMITZ, E. A.; DE FREITAS, R.. Analysis of the Polynomial Complexity of a Classical Algorithm for a Project Scheduling Problem with the Maximization of the Net Present Value. In: **30th European Conference on Operational Research (EURO)**, 2019, Dublin, Irlanda.

- LACERDA, I. M.; SCHMITZ, E. A. Valor Presente Líquido em Projetos de Software com e sem restrição de Recursos. In: **IV Escola Regional de Sistemas de Informação**, 2017, Rio de Janeiro.

8.5 PESQUISAS FUTURAS

Entre as possibilidades visualizadas para pesquisas futuras, destacam-se:

- Realizar um novo experimento que considere amostras com grafos maiores que os tratados neste experimento (acima de 320 vértices) e grafos com arestas redundantes por transitividade. Tais casos podem sujeitar os algoritmos a níveis mais elevados de estresse, com alto número de vértices e arestas.
- Decidir pela direção de busca e deslocamento ($forward$ ou $backward$) através de um modelo multicritério. Em outras palavras, os algoritmos podem considerar não somente o critério percentual de atividades com fluxo de caixa negativo para escolher a direção, mas também critérios como a topologia das redes, as posições dos vértices com fluxo de caixa negativo nas respectivas camadas, o volume global do fluxo de caixa negativo em relação ao positivo, entre outros. Tais critérios juntos e ponderados tornam a decisão da escolha da direção mais sofisticada, o que deve melhorar os resultados dos algoritmos.
- Implementar o algoritmo RSFB com algum suplemento tecnológico que permita ampliar a profundidade da pilha de recursão (para esse algoritmo) no tratamento de redes com 320 (ou mais) vértices. Em seguida, realizar um novo experimento. Com redes na ordem de 320 (ou mais) vértices, conjectura-se que o custo do algoritmo RSFB seja maior que $O(n^3)$, pois os algoritmos SAAFB e HS já apresentaram essa ordem de custo em redes com até 320 vértices neste estudo.

REFERÊNCIAS

BLAZEWICZ, J.; LENSTRA, J. K.; KAN, A. R. Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, Elsevier, v. 5, n. 1, p. 11–24, 1983. Citado na página 28.

BRUCKER, P. *Scheduling algorithms*. [S.l.]: Springer Publishing Company, 2006. Citado na página 25.

CAVALCANTE, C. C. et al. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, Elsevier, v. 112, n. 1-3, p. 27–52, 2001. Citado na página 36.

CHANG, G. J.; EDMONDS, J. The poset scheduling problem. *Order*, Springer, v. 2, n. 2, p. 113–118, 1985. Citado na página 36.

DEMEULEMEESTER, E. L.; HERROELEN, W. S. *Project scheduling: a research handbook*. [S.l.]: Springer Science & Business Media, 2006. v. 49. Citado 2 vezes nas páginas 23 e 38.

DEMEULEMEESTER, E. L.; HERROELEN, W. S.; DOMMELEN, P. V. An optimal recursive search procedure for the deterministic unconstrained max-nvp project scheduling problem. *DTEW Research Report 09603*, KU Leuven-Departement toegepaste economische wetenschappen, p. 1–15, 1996. Citado 4 vezes nas páginas 13, 20, 23 e 38.

ELMAGHRABY, S. E.; HERROELEN, W. S. The scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, Elsevier, v. 49, n. 1, p. 35–49, 1990. Citado 3 vezes nas páginas 13, 20 e 22.

ELMAGHRABY, S. E.; KAMBUROWSKI, J. The analysis of activity networks under generalized precedence relations (gprs). *Management science*, INFORMS, v. 38, n. 9, p. 1245–1263, 1992. Citado na página 28.

ERDOS, P.; RÉNYI, A. et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, Citeseer, v. 5, n. 1, p. 17–60, 1960. Citado na página 72.

FISHER, I. *The nature of capital and income*. [S.l.]: Macmillan and Cie, 1906. Citado na página 32.

FISHER, I. *Theory of interest: as determined by impatience to spend income and opportunity to invest it*. [S.l.]: Augustusm Kelly Publishers, Clifton, 1930. Citado na página 32.

FISHER, I.; BARBER, W. J. *The rate of interest*. [S.l.]: Garland Pub., 1907. Citado na página 32.

FREITAS, R. R. de. *Caracterizações e algoritmos para problemas de escalonamento*. Tese (Tese doutorado) — Universidade Federal do Rio de Janeiro, 2009. Citado na página 25.

FáVERO, L.; BELFIORE, P. *Data science for business and decision making*. [S.l.]: Academic Press, 2019. Citado 2 vezes nas páginas 74 e 77.

GITMAN, L. J.; JUCHAU, R.; FLANAGAN, J. *Principles of managerial finance*. [S.l.]: Pearson Higher Education AU, 2015. Citado na página 33.

GRAHAM, R. L. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: *Annals of discrete mathematics*. [S.l.]: Elsevier, 1979. v. 5, p. 287–326. Citado na página 28.

GRINOLD, R. C. The payment scheduling problem. *Naval Research Logistics Quarterly*, Wiley Online Library, v. 19, n. 1, p. 123–136, 1972. Citado 5 vezes nas páginas 13, 20, 21, 22 e 36.

GRÖFLIN, H.; LIEBLING, T. M.; PRODON, A. *Optimal subtrees and extensions*. [S.l.], 1982. Citado na página 36.

HERROELEN, W.; DEMEULEMEESTER, E.; REYCK, B. D. A classification scheme for project scheduling. In: *Project scheduling*. [S.l.]: Springer, 1997. p. 1–26. Citado 2 vezes nas páginas 28 e 36.

HERROELEN, W. S.; DOMMELEN, P. V.; DEMEULEMEESTER, E. L. Project network models with discounted cash flows a guided tour through recent developments. *European Journal of Operational Research*, Elsevier, v. 100, n. 1, p. 97–121, 1997. Citado 3 vezes nas páginas 16, 17 e 20.

HERROELEN, W. S.; GALLENS, E. Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, Elsevier, v. 65, n. 2, p. 274–277, 1993. Citado 3 vezes nas páginas 13, 20 e 22.

JUANG, S.-H. *Optimal solution of job shop scheduling problems: A new network flow approach*. [S.l.]: The University of Iowa, 1994. Citado na página 36.

KAZAZ, B.; SEPIL, C. Project scheduling with discounted cash flows and progress payments. *Journal of the Operational Research Society*, Taylor & Francis, v. 47, n. 10, p. 1262–1272, 1996. Citado 3 vezes nas páginas 13, 20 e 22.

NELDER, J. A.; WEDDERBURN, R. W. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, Wiley Online Library, v. 135, n. 3, p. 370–384, 1972. Citado na página 74.

PINEDO, M. L. *Scheduling*. [S.l.]: Springer, 2012. v. 29. Citado 2 vezes nas páginas 25 e 26.

PMI. *A Guide to the Project Management Body of Knowledge*. [S.l.]: Project Management Institute, 2017. Citado na página 25.

PMI. *Practice Standard for Work Breakdown Structure*. [S.l.]: Project Management Institute, 2019. v. 3rd edition. Citado na página 25.

POPPER, K. R. *A lógica da pesquisa científica*. [S.l.]: Editora Cultrix, 2004. Citado 3 vezes nas páginas 19, 66 e 67.

ROUNDY, R. O. et al. A price-directed approach to real-time scheduling of production operations. *IIE transactions*, Taylor & Francis, v. 23, n. 2, p. 149–160, 1991. Citado na página 36.

RUSSELL, A. Cash flows in networks. *Management Science*, INFORMS, v. 16, n. 5, p. 357–373, 1970. Citado 6 vezes nas páginas 13, 16, 20, 21, 22 e 34.

SANKARAN, J. K.; BRICKER, D. L.; JUANG, S.-H. A strong fractional cutting-plane algorithm for resource-constrained project scheduling. *International Journal of Industrial Engineering*, PUBLISHING HORIZONS, v. 6, p. 99–111, 1999. Citado na página 36.

SCHWINDT, C.; ZIMMERMANN, J. A steepest ascent approach to maximizing the net present value of projects. *Mathematical Methods of Operations Research*, Springer, v. 53, n. 3, p. 435–450, 2001. Citado 5 vezes nas páginas 13, 17, 20, 23 e 41.

SMITH-DANIELS, D. E. Summary measures for predicting the net present value of a project. In: IN COLLEGE OF ST. THOMAS ST. PAUL. *Minnesota Working Paper*. [S.l.], 1986. Citado 3 vezes nas páginas 13, 20 e 22.

VANHOUCKE; DEMEULEMEESTER, E.; HERROELEN, W. *On maximizing the net present value of a project under resource constraints*. K.U.Leuven - Departement toegepaste economische wetenschappen, 1999. 1–24 p. Disponível em: https://lirias.kuleuven.be/retrieve/60236. Citado 5 vezes nas páginas 13, 17, 20, 23 e 38.

VANHOUCKE, M. An efficient hybrid search algorithm for various optimization problems. In: SPRINGER. *European Conference on Evolutionary Computation in Combinatorial Optimization*. [S.l.], 2006. p. 272–283. Citado 6 vezes nas páginas 13, 17, 20, 24, 48 e 53.

VANHOUCKE, M.; DEMEULEMEESTER, E.; HERROELEN, W. A validation of procedures for maximizing the net present value of a project. *DTEW Research Report 0030*, KU Leuven; Leuven, 2000. Citado 2 vezes nas páginas 23 e 41.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. [S.l.]: Elsevier, 2009. v. 2. Citado na página 66.

WILSON, J. M. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, Elsevier, v. 149, n. 2, p. 430–437, 2003. Citado na página 26.