

Sobre a complexidade de um algoritmo clássico para escalonamento de projetos com maximização de valor presente líquido

Isac M. Lacerda¹, Jayme L. Szwarcfiter¹, Eber A. Schmitz¹, Rosiane de Freitas²

¹Programa de Pós-Graduação em Informática - Universidade Federal do Rio de Janeiro (PPGI/UFRJ)

²Instituto de Computação - Universidade Federal do Amazonas (IComp/UFAM)

isac.mendes@gmail.com, {jayme, eber}@nce.ufrj.br,
rosiane@icom.ufam.edu.br

RESUMO

Entre os problemas de escalonamento de projetos está a maximização de Valor Presente Líquido (VPL) sem restrição de recursos, com precedência do tipo término-início e *lag* zero entre as atividades. Para esse problema, Demeulemeester et al. [1996] propuseram um algoritmo organizado em três passos. De acordo com tais autores, o primeiro e o segundo passos ocorrem em tempo de complexidade $O(n^2)$. No entanto, a complexidade do terceiro e mais importante dos passos era uma questão em aberto até então, impossibilitando determinar a complexidade total do algoritmo. Assim, neste trabalho, apresentamos uma análise da complexidade para esse passo, onde o pior caso ocorre em um arranjo com atividades sequenciais, que exige o máximo de desconexões e conexões entre as atividades. Deste modo, o terceiro passo também se configura em tempo de complexidade $O(n^2)$. Tal caracterização confirma a robustez da solução proposta, bem como contribui para outros algoritmos e trabalhos de pesquisa relacionados.

PALAVRAS CHAVE. Escalonamento de Projetos, Maximização de VPL, Sem Restrição de Recursos.

ABSTRACT

Among the project scheduling problems is the maximization of the Net Present Value (NPV) without resources constrained, with finish-start precedence and zero-lag between activities. For this problem, Demeulemeester et al. [1996] proposed an algorithm organized in three steps. Given by that authors, the first and second steps happen in $O(n^2)$ time complexity. However, the complexity of the third and most important step was an open question so far, making it impossible to determine the overall complexity of the algorithm. Thus, in this work, we present the analysis of the computational complexity to this step, where the worst case occurs in an arrangement with sequential activities, where requires the maximum of disconnections and connections of activities. That way, the third step also configures an $O(n^2)$ time complexity. Such characterization confirms the robustness of the solution proposed, as well as contributes to other related algorithms and work researches.

KEYWORDS. Project Scheduling, Net Present Value Maximization, Unconstrained Resource.

1. Introdução

A avaliação econômico-financeira de um projeto pode ser realizada com apoio de diferentes métodos. Entre as opções mais conhecidas está o método do Valor Presente Líquido (VPL). A abordagem do VPL inclui a determinação em valor presente para pagamentos futuros [Lacerda e Schmitz, 2017] e, desta maneira, uma taxa mínima de atratividade deve ser considerada. Em outras palavras, encontrar o VPL para um projeto envolve conhecer o investimento inicial, o fluxo de caixa em data específica e o custo do capital definido pelos patrocinadores.

Aparentemente fazer o cálculo de Valor Presente Líquido (VPL) de um projeto é uma tarefa simples, desde que os dados estejam disponíveis. No entanto, a busca por maximização de ganhos monetários passa por não só identificar os dados, mas também estabelecer as melhores condições para a realização do projeto. Por conta disso, é necessário avaliar, por exemplo, quais são os possíveis ordenamentos válidos para as atividades. Ordenamentos válidos, nesse caso, são restrições conhecidas e atendidas. Projetos têm várias atividades com diferentes restrições de precedência entre elas e, por isso, muitas combinações se tornam factíveis. Muitas combinações de realização fazem com que diferentes VPLs sejam possíveis. Assim, encontrar o cenário que maximiza o VPL passa a ser uma preocupação natural. Nesse sentido, Russell [1970] é considerado o pioneiro com o trabalho denominado *Cash Flow in Networks*.

De modo geral, problemas que envolvem ordenamento ótimo, atendendo a restrições, são denominados de problemas de escalonamento [Rodrigues, 2009]. Como existem muitas classes diferentes desses problemas, muitos algoritmos têm sido propostos [Brucker, 2006]. Pinedo [2008] apresenta a importância desse conceito, destacando que escalonamento não é só um conjunto de técnicas, mas um modo de tomada de decisão crucial para produção e desenvolvimento de serviços industriais. Assim, escalonamento de projetos, como uma aplicação especializada, refere-se ao tratamento de significativos problemas para organizações. Mas escalonamento é motivo de interesse não só para o mercado. Nas últimas décadas, muitos trabalhos científicos foram desenvolvidos envolvendo esse tema. Uma visão geral sobre a literatura de escalonamento de projetos para maximização de VPL até 1997 pode ser verificada no trabalho de Herroelen et al. [1997] e uma versão mais recente pode ser verificada no trabalho de Leyman e Vanhoucke [2016]. Mais detalhes sobre essas duas revisões são descritos na seção denominada Trabalhos Relacionados.

Nesse contexto, tomou-se como objeto de estudo um clássico algoritmo (para escalonar atividades de projeto a fim de maximizar o VPL), proposto originalmente por Demeulemeester et al. [1996], refinado por Vanhoucke et al. [1999] e discutido em detalhes por Demeulemeester e Herroelen [2002]. O algoritmo despreza restrição de recursos, mas considera dependências entre atividades do tipo término-início. Nesse algoritmo, que fornece uma solução exata, segundo seus autores, três passos são propostos, mas apenas os dois primeiros tiveram classificação de complexidade de pior caso definida. O terceiro passo, deste modo, sem classificação de complexidade definida, passou a ser o ponto central de investigação. Com isso, a partir da realização de séries de avaliações, constatou-se que o terceiro passo, assim como os dois primeiros, também é $O(n^2)$.

Por fim, a estrutura deste artigo, além desta introdução, inclui as seções: Formulação do Problema; Trabalhos Relacionados; Algoritmo Principal; Séries de Avaliações; Definição da Complexidade; e Conclusões.

2. Formulação do Problema

Graham et al. [1979] e Blazewicz et al. [1983] propuseram uma abordagem conhecida como notação de 3-campos para expressar problemas de escalonamento de máquinas. Posteriormente, motivados pela grande variedade de problemas de escalonamento em projetos, Demeulemeester et al. [1997] introduziram uma extensão na notação original de 3-campos, adaptando-a ao contexto de escalonamento e ao vocabulário de projetos. Na abordagem estendida para projetos, os campos, $\alpha \mid \beta \mid \gamma$, denotam: α características dos recursos (com até três subcampos); β características das atividades (com até oito subcampos); e γ critérios de otimalidade.

De forma geral, o problema analisado neste trabalho envolve maximizar o VPL de projeto com restrições de precedência entre atividades, mas sem restrições de recursos. As restrições de precedência entre as atividades, no entanto, limitam-se ao tipo término-início e com *lag* zero. As atividades do projeto têm fluxo de caixa conhecido, bem como o projeto tem uma data de fim imposta

conhecida. Diante da notação estendida de 3-campos, o mesmo problema pode ser exposto da seguinte forma: ${}^{\circ} \mid cpm, \delta_n, c_j \mid max-npv$. Onde: ${}^{\circ}$ indica a ausência de restrição de recursos; *cpm* indica que as atividades estão sujeitas estritamente ao tipo de precedência término-início com *lag* zero; δ_n indica que há uma data limite de fim imposta ao projeto (*deadline*); c_j indica que as atividades têm fluxo de caixa conhecido; e *max-npv* (*maximization of the net present value*) indica que o Valor Presente Líquido (VPL) deve ser maximizado.

A formulação pode contar ainda com a representação gráfica conhecida como Seta No Nó, conforme Figura 1. Nessa notação, pode-se assumir a existência de um grafo $G = (V, E)$, onde V representa os vértices (atividades) e E representa as arestas (precedências entre as atividades). No conjunto V de vértices existem dois deles destinados respectivamente às marcações de início e de fim no grafo (no inglês, denominados como *dummies*). Além disso, os vértices seguem uma orientação crescente de numeração, em relação às precedências. A duração de cada atividade i (vértice) pode ser denominada como d_i e a realização de cada atividade envolve um fluxo de caixa de despesa (negativo) ou receita (positivo) ao longo da duração. É importante dizer, que os vértices de marcações de início e fim (*dummies*), têm duração zero e fluxo de caixa zero. Com isso, o modelo para o problema (${}^{\circ} \mid cpm, \delta_n, c_j \mid max-npv$) tem como função objetivo a expressão (1):

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

Onde, c_i indica o valor do fluxo de caixa de uma atividade i (no seu tempo de término), enquanto s_i refere-se ao início de uma atividade, d_i refere-se à duração de uma atividade e $e^{-\alpha}$, que é igual a $1/(1+r)$, refere-se ao fator de desconto, sendo r a taxa de desconto. Além disso, a expressão (1) está sujeita às seguintes restrições:

$$s_i + d_i \leq s_j; \forall (i, j) \in E \quad (2)$$

$$s_1 = 0 \quad (3)$$

$$s_n \leq \delta_n \quad (4)$$

$$s_i \in \mathbb{N}; i = 2, 3, 4, \dots, n \quad (5)$$

Portanto, no modelo, a expressão (1) tem como objetivo alcançar o maior VPL possível. A expressão (2) restringe que o início de qualquer atividade (s_i) mais sua duração (d_i) seja menor ou igual ao início de qualquer uma de suas sucessoras (s_j) (válido para toda aresta do conjunto E). A expressão (3) indica que o início da atividade 1 deve ser no tempo zero. A expressão (4) indica que o início da última atividade (s_n) deve ser menor ou igual à data de fim limite imposta ao projeto (δ_n). Por último, a expressão (5) garante que o início das atividades deve pertencer ao conjunto dos números naturais.

Importante destacar que, para este problema, alguns autores já demonstraram que é possível ter uma solução em tempo polinomial [Cavalcante et al., 1998], [Chang e Edmonds, 1985], [De Souza e Wolsey, 1997], [Gröflin et al., 1982], [Juang, 1994], [Margot et al., 1990], [Roundy et al., 1991] e [Sankaran et al., 1999]. Além disso, embora o problema tenha sido aqui apresentado como não linear, Grinold [1972] demonstra que é possível transformá-lo em um problema de programação linear, assumindo uma visão orientada a eventos (fluxo de caixa com eventos associados à rede de precedência). Modelado com programação linear, o problema é chamado por Dantzig [1963] como problema de distribuição ponderada.

3. Trabalhos Relacionados

Em 1997 Herroelen et al. [1997] publicaram um artigo denominado por *Project Network Models with Discounted Cash Flows a Guided Tour Through Recent Developments*. Nesse artigo, os autores destacam que, até a ocasião, o maior interesse sobre escalonamento de projetos esteve voltado para a minimização da duração dos projetos (*makespan*). Por conta disso, o interesse sobre aspectos financeiros no escalonamento estava apenas em estágio inicial. Deste modo, propuseram o seu trabalho como uma revisão crítica de outras pesquisas que tentavam capturar objetivos monetários e financeiros em problemas de *max-npv*. Diante disso, Herroelen et al. [1997] apresentam trinta e quatro trabalhos agrupados em seis principais categorias de soluções. Tais categorias são denominadas como: a) Soluções Determinísticas sem Restrição de Recursos; b) Soluções Estocásticas sem Restrição de Recursos; c) Soluções com Restrição de Recursos Renováveis; d) Soluções com Restrição de Recursos Não-Renováveis; e) Soluções com Restrição de Recursos Renováveis e Não-Renováveis; f) Soluções de Escalonamento de Pagamentos. Embora o trabalho de Herroelen et al. [1997] apresente características gerais das soluções catalogadas e faça alguma comparação entre elas, não são destacadas características de complexidade de melhor caso, pior caso ou caso médio dos algoritmos propostos. Vale dizer ainda que das seis categorias, a primeira representa a de interesse principal deste artigo.

Em 2016 Leyman e Vanhoucke [2016] publicaram um artigo denominado por *Capital- and Resource-Constrained Project Scheduling with Net Present Value Optimization*. Nesse artigo, os autores apresentam o problema de escalonamento de projetos com restrição de capital com fluxo de caixa descontado (P1), assim como o problema de escalonamento de projetos com restrição de capital e também recursos com fluxo de caixa descontado (P2). Entre as contribuições desse trabalho, tem-se a proposta de três modelos de fluxo de caixa diferentes na qual afetam a disponibilidade de capital durante a execução do projeto. Outra contribuição envolve aspectos de escalonamentos para melhorar a disponibilidade de capital para P1 e também para P2. Esses aspectos de escalonamento foram implementados como parte de metaheurísticas da literatura, permitindo comparações de desempenho entre elas. Para isso, os autores fizeram uma retrospectiva da literatura destacando a origem do interesse por *max-npv*, bem como uma apresentação com a ramificação de temas que envolvem o capital de um projeto como uma possível restrição de escalonamento. Leyman e Vanhoucke [2016] fazem distinção entre três grandes áreas de pesquisa com relação ao fluxo de caixa de entrada das atividades, que dependem de negociações entre cliente e fornecedor. Essas áreas são denominadas como “*Timing*”, “*Tamanho*” e “*Timing e Tamanho*”. Sob tal classificação, Leyman e Vanhoucke [2016] apresentaram uma compilação da literatura com vinte e dois trabalhos, incluindo o deles. No entanto, não são discutidos aspectos formais sobre a complexidade dos algoritmos, assim como no trabalho de Herroelen et al. [1997].

Em 1996, Demeulemeester et al. [1996] publicaram um algoritmo denominado *An Optimal Recursive Search Procedure*. Esse mesmo algoritmo foi novamente apresentado, mas com mais detalhes, no trabalho de Demeulemeester e Herroelen [2002]. Nessa última publicação, além da versão original, três variações do algoritmo são discutidas e comparadas. Deste modo, as quatro próximas subseções anunciam brevemente a existência de tais algoritmos.

3.1. Algoritmo 1: Original Recursive Search Procedure

O *Recursive Search Procedure* é o principal algoritmo entre os quatro da sequência. Isso se deve a duas razões essenciais. Primeiro porque essa se refere à solução original (entre quatro possíveis soluções, sugeridas pelos mesmos autores, para o mesmo problema). Segundo por deixar em aberto uma importante lacuna teórica, no que se refere à definição de complexidade do último e mais importante dos seus passos. A seção 4 deste trabalho descreve com mais detalhes os três passos desse algoritmo.

3.2. Algoritmo 2: Forward-Backward Recursive Search Procedure

A realização de experimentos com o algoritmo original da seção anterior permitiu identificar o forte impacto que os vértices com fluxo de caixa negativo podem causar. Em outras palavras, quanto mais vértices com fluxo de caixa negativo, mais esforço é exigido do algoritmo. Essa constatação inspirou Vanhoucke et al. [2000] a fazerem uma implementação reversa do algoritmo original, quando mais de 50% dos vértices tivessem valor de fluxo de caixa negativo. Nesse caso, o *Recursive Search Procedure* começa atrasando os vértices do grafo ao invés de adiantar. Assim, as atividades são agendadas o mais tarde possível, usando o vértice final como base ao invés do vértice inicial. Assim, o algoritmo busca por conjuntos de atividades que devem ser deslocadas para trás para aumentar o VPL.

3.3. Algoritmo 3: Steepest Ascent Procedure

Os autores Schwindt e Zimmermann [2001] implementaram um algoritmo denominado *Steepest Ascent Procedure* (SAP) para *max-npv* em grafos de projetos sujeitos a relações de precedência generalizadas com intervalos temporais mínimos e máximos. Embora a mecânica do SAP seja diferente da proposta original do *Recursive Search Procedure*, segundo Demeulemeester e Herroelen [2002], o SAP pode ser prontamente adaptado ao problema com relações de precedência da mesma discussão. Deste modo, destacam algumas semelhanças de estratégias entre os dois algoritmos. A partir de uma árvore de abrangência correspondente a um agendamento que antecipa o quanto possível cada um dos vértices, ambos os procedimentos (SAP e *Recursive Search Procedure*) identificam iterativamente subárvores que devem ser atrasadas por um intervalo de tempo máximo permitido pelas restrições de precedência. No *Recursive Search Procedure*, são utilizadas etapas recursivas únicas. Já no SAP, são utilizadas iterações e seleções para identificar a direção de subida mais íngreme (*Steepest Ascent*) e as iterações subsequentes para ascender ao próximo vértice correspondente a um novo escalonamento. Embora o algoritmo SAP adaptado tenha sido apresentado na publicação de Demeulemeester e Herroelen [2002], discussões de complexidade não são tratadas.

3.4. Algoritmo 4: Recursive Search Procedure Revised

Uma versão revisada do algoritmo *Recursive Search Procedure* foi proposta por Vanhoucke et al. [2000]. Nessa versão, o algoritmo divide a lógica do passo 3 original em duas partes. A primeira parte (identificada como a etapa *RECURSION*) procura apenas conjuntos de atividades com fluxo de caixa descontado negativo, que, como resultado, são candidatos a serem atrasados. Os conjuntos de atividades, denominados *SA* (*Set Activities*) encontrados e salvos nesta etapa, são indicados como $SS = SA$. A segunda etapa, chamada de etapa *SHIFT ACTIVITIES* calcula um intervalo de deslocamento permitido para os conjuntos de atividades visitados. Embora Vanhoucke et al. [2000] tenham conseguido melhores resultados empíricos e tenham proposto um código aparentemente mais enxuto, o algoritmo original se tornou o objeto de estudo principal pelas razões mencionadas na seção 3.1.

4. Algoritmo Principal

Como anunciado, o algoritmo original de Demeulemeester e Herroelen [2002] está organizado em três passos. Todos os três passos utilizam um grafo denominado de $G = (V, E)$, em que os vértices denotam todas as atividades e as arestas denotam todas as precedências (sempre do tipo término-início). O grafo G funciona como um gabarito de verificações para os três passos e não sofre alterações. Um exemplo de grafo G é o da Figura 1, onde os vértices 1 e 14 são respectivamente as marcações de início e fim. Já os valores acima dos vértices representam duração, enquanto os valores abaixo representam os respectivos fluxos de caixa. Deste modo, usando o vértice 2 como exemplo, tem-se uma atividade com duração de 6 dias e fluxo de caixa de -140 unidades de dinheiro.

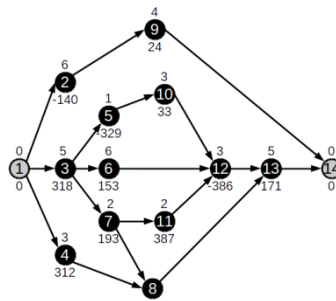


Figura 1: Exemplo de um Grafo G .

4.1. Passo 1

O primeiro passo do algoritmo usa o grafo G para verificar as precedências das atividades e montar uma estrutura chamada *Árvore Cedo* (no original, *Early Tree*). Nessa árvore, calcula-se o término mais cedo possível para cada atividade e se cria arestas entre as atividades com base no método do caminho crítico (CPM – *Critical Path Method*). Ainda no primeiro passo, cria-se uma aresta conectando o vértice final ao vértice inicial, útil para a busca recursiva do terceiro passo (aresta entre os vértices 14 e 1, na Figura 2). Ao executar o passo 1, tomando o grafo G da Figura 1 como gabarito, tem-se como resultado a *Árvore Cedo* da Figura 2. As linhas tracejadas da Figura 2 indicam que essas arestas não existem na *Árvore Cedo*, embora existam no grafo G , servindo como fonte de verificação. Para Demeulemeester e Herroelen [2002], este passo é $O(n^2)$.

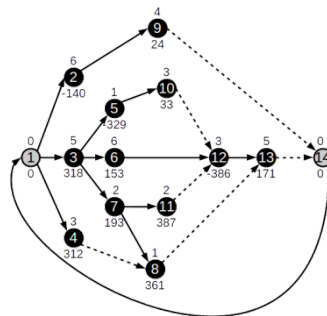


Figura 2: *Árvore Cedo* para o passo 1 e *Árvore Corrente* para o passo 2.

4.2. Passo 2

O segundo passo do algoritmo constrói a *Árvore Corrente* (no original, *Current Tree*), inicialmente fazendo cópia da *Árvore Cedo*. Nesse passo, as atividades são varridas em ordem decrescente de modo a encontrar alguma atividade com fluxo de caixa negativo e sem sucessão. Caso alguma atividade seja encontrada nessa condição, tal atividade é atrasada o quanto possível na *Árvore Corrente*. Como na *Árvore Corrente*, deste caso, não existem vértices com valores negativos e sem sucessores, o resultado do segundo passo será a mesma árvore da Figura 2. Para Demeulemeester e Herroelen [2002], o segundo passo, assim como o primeiro, também é $O(n^2)$.

4.3. Passo 3

O terceiro passo, do *Recursive Search Procedure*, como mencionado anteriormente, não recebeu classificação de complexidade. Por conta disso, é o objeto principal de análise. De modo geral, o passo 3 identifica e atrasa conjuntos de atividades que podem aumentar o VPL. Para isso, dinamicamente, são criadas e excluídas precedências na *Árvore Corrente*. A execução do passo 3 começa a partir do vértice 1, a fim de identificar conjuntos de atividades passíveis de deslocamento em direção à data de fim imposta ao projeto, ou seja, em oposição ao vértice 1. Um conjunto de atividades é candidato

a deslocamento, quando a soma do fluxo de caixa de suas atividades tem valor negativo. Vale dizer também que deslocamentos em direção ao instante zero (vértice 1), não são capazes de aumentar o VPL, já que o passo 1, com a *Árvore Cedo*, fez toda a antecipação possível de cada atividade, respeitando as precedências do grafo G . Quando um conjunto de atividades que pode aumentar o VPL é encontrado, a *Árvore Corrente* é alterada. Nesse caso, é feita uma exclusão da aresta que antecede o conjunto de atividades e uma inclusão de aresta entre a última atividade do conjunto que deve ser deslocado até a atividade sucessora mais próxima (de acordo com o grafo G). Com isso, as atividades deslocadas têm suas datas de início e fim recalculadas em função da atividade sucessora. A varredura ocorre de forma crescente e recursiva e termina quando não se encontra um conjunto de atividades que pode aumentar o VPL. De modo a favorecer a legibilidade, uma versão do pseudocódigo original foi reescrita, preservando a lógica, e está apresentada abaixo.

Quadro 1: Pseudocódigo do passo 3 reescrito.

```

01: //Elementos globais
02: SA: conjunto //Conjunto de atividades candidato a deslocamento (Set of Activities)
03: CA: conjunto //Atividades visitadas em uma chamada ao procedimento PASSO_3 () (Considered Activities)
04: DC: ponto flutuante //Fluxo de caixa descontado (Discounted Cash Flow)
05: procedimento PASSO_3 () //Inicia a varredura sempre a partir do vértice 1
06:   CA ← ∅
07:   SA', DC' ← RECURSÃO (1)
08:   escreva (SA', DC')
09: função RECURSÃO (vértice): SA', DC' //Visita os vértices para decidir sobre as podas
10:   SA ← vértice
11:   DC ← DCF (vértice) //Função auxiliar que calcula o fluxo de caixa do vértice corrente
12:   CA ← CA união vértice //Inclui o vértice corrente em CA
13:   paraTodo (i em SUCC (vértice) e (i ∉ CA)) faça //Varre todos os sucessores do vértice corrente
14:     SA', DC' ← RECURSÃO (i) //Nova chamada da função RECURSÃO () para um sucessor
15:     se (DC' ≥ 0) então //SA e DC são incrementados, p/ DC' positivo
16:       SA ← SA união SA'
17:       DC ← DC + DC'
18:     senão
19:       PODA (vértice, SUCC (vértice)) //Função auxiliar que poda um conjunto SA
20:       ENXERTA (SA) //Função auxiliar que enxerta SA
21:       PASSO_3 () //Reinicia a varredura no grafo
22:   paraTodo (i em PRED (vértice) e (i ∉ CA)) faça //Varre todos os predecessores do vértice corrente
23:     SA', DC' ← RECURSÃO (i) //Nova chamada da função RECURSÃO () para um predecessor
24:     SA ← SA união SA' //Incrementa SA
25:     DC ← DC + DC' //Incrementa DC
26:   retorna (SA, DC)

```

O pseudocódigo do Quadro 1 apresenta três elementos globais: SA (no inglês, *Set of Activities*), conjunto de atividades candidato a deslocamento se o seu fluxo de caixa descontado for negativo; CA (no inglês, *Considered Activities*), conjunto de atividades visitadas em uma sequência de chamadas à função RECURSÃO () dentro de uma chamada ao procedimento PASSO_3 (); e DC (no inglês, *Discounted Cash Flow*), fluxo de caixa descontado de uma ou mais atividades visitadas em uma chamada ao PASSO_3 (). O pseudocódigo tem um único procedimento e uma função principal, que são chamados recursivamente. Na linha 5, o procedimento é denominado de PASSO_3 () e é utilizado para iniciar uma varredura na *Árvore Corrente*, sempre a partir do vértice 1. Já a função, na linha 09, é denominada de RECURSÃO () e é utilizada para avaliar se um conjunto de atividades tem saldo negativo de fluxo de caixa para, então, fazer deslocamentos. A função RECURSÃO () recebe uma atividade como argumento e retorna SA' e DC', que representam conjuntos de sucessores ou predecessores visitados e seus respectivos fluxos de caixa descontados. RECURSÃO () é composta por dois laços. O primeiro

laço verifica sucessores de uma atividade (vértice) não visitada em uma chamada de `PASSO_3()` e o segundo laço verifica predecessores da mesma atividade também em uma chamada do `PASSO_3()`.

A execução do algoritmo deste passo tem como entrada a *Árvore Corrente* no estado indicado da Figura 2. Ao final da execução, a *Árvore Corrente* passa a ter o estado da Figura 3. Além disso, a Figura 4 representa um esquema de empilhamento das chamadas de `PASSO_3()` e `RECURSÃO()`, destacando os conjuntos de vértices SA desconectados de predecessores e conectados em sucessores (linhas pontilhadas).

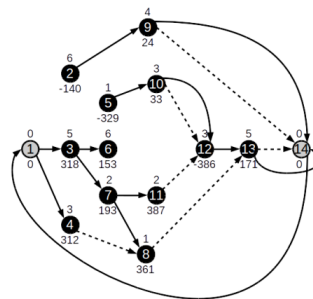


Figura 3: *Árvore Corrente* totalmente alterada pelo passo 3.

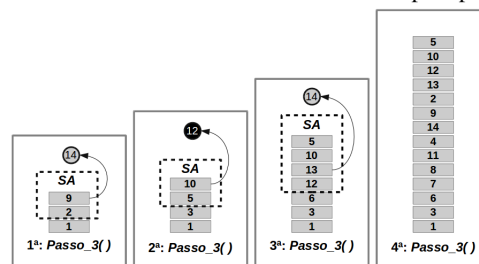


Figura 4: Empilhamento das chamadas de `PASSO_3()` e `RECURSÃO()`.

5. Séries de Avaliações

Sob o caráter exploratório, as avaliações de comportamento do algoritmo foram organizadas em três séries distintas. Alguns dos termos utilizados, com o intuito de facilitar a compreensão das ações do algoritmo, estão listados a seguir:

- **Árvore:** uma árvore m -ária T , $m \geq 2$, é um conjunto finito de elementos, denominados vértices, tais que se:
 - $T = \emptyset$, a árvore é dita vazia, ou;
 - Contém um vértice especial r , chamado raiz de T , e os restantes podem ser sempre divididos em m subconjuntos disjuntos, as i -ésimas subárvores de r , $1 \leq i \leq m$, as quais são também árvores m -árias [Szwarcfiter e Markenzon, 2010];
- **Subárvore:** Conjunto disjunto de vértice em uma árvore, conectado a um vértice raiz [Szwarcfiter e Markenzon, 2010];
- **Poda:** Exclusão de aresta entre dois vértices na *Árvore Corrente*.
- **Enxerto em nível intermediário:** Inclusão de aresta entre dois vértices na *Árvore Corrente*, sendo que o vértice que recebe a seta não pode ser o vértice final ou ser um predecessor direto ou indireto do vértice final. Na Figura 3, a linha cheia entre os vértices 10 e 12 representa um exemplo de enxerto em nível intermediário.

- **Enxerto no fim:** Inclusão de aresta entre dois vértices na *Árvore Corrente*, sendo que o vértice que recebe a seta já deve ter sido enxertado junto ao vértice final, direta ou indiretamente (como indica a linha cheia entre os vértices 9 e 14 na Figura 3).
- **Vértice Negativo Suficientemente Grande (VNSG):** Vértice com valor de fluxo de caixa negativo, mesmo quando somado a valores de fluxo de caixa de eventuais vértices sucessores e/ou predecessores que compõem uma subárvore na *Árvore Corrente*.

5.1. Características e Resultados Gerais das Séries

Na primeira série de avaliações, foram estruturadas arbitrariamente árvores com apenas onze vértices (incluindo os vértices de início e de fim). A realização da primeira série revelou que as árvores mais altas com VNSG com maior proximidade do vértice final exigiram maior esforço do algoritmo. Na segunda série de avaliações, foram estruturadas árvores com números variados de atividades, mas com um arranjo padronizado, destacando três filas paralelas de vértices. Na segunda série, os vértices negativos foram posicionados em pontos mais próximos possível ao vértice inicial e apresentaram comportamento empírico em ordem quadrática, à medida que novos vértices foram inseridos. Na terceira e última série de avaliações, foram estruturadas árvores que testaram o esforço do algoritmo, comparando podas seguidas de enxerto no fim com enxerto em nível intermediário. Entre as estruturas testadas, a árvore com todos os vértices em sequência foi a que exigiu maior esforço do algoritmo.

6. Definição da Complexidade

As três séries serviram para explorar o comportamento do algoritmo. Desta maneira, seus resultados ajudaram a estruturar cinco lemas, um teorema e as respectivas demonstrações (com a definição de complexidade computacional do passo em aberto).

Lema 6.1. *O processo recursivo de PASSO_3 () termina.*

Demonstração: *O empilhamento das chamadas ao procedimento PASSO_3 () ocorre através da linha 21 do pseudocódigo no Quadro 1. Essa linha é executada sempre que o fluxo de caixa retornado por uma chamada de RECURSÃO () for negativo para um vértice com sucessor não visitado até então. Deste modo, quando não há vértices negativos, a linha 21 nunca será executada. Além disso, mesmo que todos os vértices sejam visitados, CA, na linha 13, funciona como uma memória de visita que garante, em uma chamada específica ao PASSO_3 (), que apenas vértices não visitados com valor negativo de fluxo de caixa empilhem chamadas desse procedimento, através da linha 21. Portanto, o processo termina.*

Lema 6.2. *O processo recursivo de RECURSÃO () termina.*

Demonstração: *O empilhamento das chamadas de RECURSÃO () ocorre através das linhas 14 e 23 do pseudocódigo no Quadro 1, sempre dentro de uma chamada ao PASSO_3 (). Essas linhas estão dentro de laços que têm como critério de entrada a não visitação prévia do vértice, memorizada no conjunto CA ($i \notin CA$). Portanto, para que o processo fosse infinito deveria haver repetição de visita aos vértices, o que é impossível com as verificações realizadas através do mesmo critério de entrada nos laços, conforme linhas 14 e 23. Em outras palavras, o conjunto CA garante que um vértice específico é visitado exclusivamente na condição de sucessor ou predecessor, nunca nas duas condições, em uma chamada ao procedimento PASSO_3 ().*

Lema 6.3. *Em uma chamada de PASSO_3 () os vértices são visitados no máximo uma vez.*

Demonstração: *As linhas 14 e 23 do pseudocódigo no Quadro 1 garantem que um vértice só é visitado uma vez em uma chamada ao procedimento PASSO_3 (), pois a memória do conjunto CA sofre apenas incrementos e não é reiniciada dentro dessa chamada, conforme linha 12.*

Tabela 1: Visibilidade dos Laços.

Laço	Descrição	Cenários				
		1°	2°	3°	...	n°
1°	Vértices Visíveis à Frente	$(n - 1)$	$(n - 2)$	$(n - 3)$	...	2
2°	Vértices Visíveis para Trás	1	2	3	...	$(n - 2)$
Total		n	n	n	...	n

Lema 6.4. A soma do número máximo de vértices visíveis ao primeiro e ao segundo laços sempre é n vértices (todos os vértices), em uma única chamada ao PASSO_3(). Quando uma poda ocorre, a visibilidade do primeiro laço diminui e a visibilidade do segundo laço aumenta, na medida da poda, para uma nova chamada ao procedimento PASSO_3().

Demonstração: O conjunto CA é definido como vazio, sempre que uma nova chamada ao PASSO_3() é realizada, conforme linha 6. Por outro lado, CA sofre incrementos (linha 12), quando RECURSÃO() é executada, em uma mesma chamada ao PASSO_3(). Deste modo, a visibilidade máxima do primeiro laço é $n - 1$ vértices (todos os vértices menos o vértice final), assumindo que todos os vértices possíveis foram deslocados, pelo passo 1, para as datas mais cedo possível. Nesse mesmo cenário, a visibilidade do segundo laço é apenas 1 vértice, o vértice final. Quando uma poda ocorre, 1 ou mais vértices são desconectados de um predecessor e esse conjunto é conectado a um sucessor mais próximo. Isso faz com que, em uma nova chamada ao procedimento PASSO_3(), a perspectiva de sucessores tenha diminuído e a perspectiva de predecessores tenha aumentado, na medida da poda. Portanto, a relação entre as visibilidades de vértices para o primeiro e para o segundo laços pode ser expressa por duas progressões aritméticas, sendo a primeira decrescente e a segunda crescente. Assim, assumindo que serão feitas podas de um único vértice (a fim de maximizar o número de podas), a razão de ambas as progressões tem o valor de 1 vértice. Portanto, somar os elementos par a par das progressões sempre gera n vértices. A Tabela 1 destaca os possíveis cenários válidos de visibilidade para os dois laços.

Lema 6.5. Se não houver poda em uma varredura através de PASSO_3(), o algoritmo termina.

Demonstração: A poda ocorre quando o fluxo de caixa de um conjunto de atividades (uma ou mais) é negativo, conforme teste condicional da linha 15, dentro do primeiro laço. Sabendo que o único ponto de recomeço de varredura através de PASSO_3() está dentro do primeiro laço de RECURSÃO() (executado para vértices sucessores não previamente visitados) na linha 21, se os vértices retornarem fluxos de caixa positivo ou forem avaliados como vértices predecessores (dentro do segundo laço), não haverá recomeço de varredura através de PASSO_3(). Portanto, cada um dos vértices será visitado uma única vez (na condição exclusiva de sucessor ou predecessor) e o algoritmo termina o trabalho.

Teorema 6.6 (Geral). O número máximo de visitas aos vértices não excede à soma das visibilidades (passos à frente e passos para trás) para os dois laços componentes do algoritmo. Assim, o esforço do algoritmo é no máximo $O(n^2)$.

Demonstração: Cinco deduções compõem a prova do teorema. **A primeira é:** o número máximo de podas é $n - 2$ vértices (todos os vértices menos os de início e fim). **A segunda é:** Como em cada varredura, através de PASSO_3(), um vértice é visitado no máximo uma vez, o número máximo de visitas em uma varredura é n , considerando os vértices de início e fim. **A terceira é:** se todos os vértices forem visitados em uma chamada ao procedimento PASSO_3(), sem uma poda, o algoritmo termina o seu trabalho. **A quarta é:** se houver uma poda, a visibilidade do primeiro laço diminui

e a visibilidade do segundo laço aumenta, na perspectiva da próxima chamada ao procedimento PASSO_3 (). Esse processo pode se repetir $n - 2$ (número máximo de podas). Deste modo, o cenário que tem o maior número de podas possível é o indicado como primeiro da Tabela 1 (primeira coluna no quadrante de cenários), onde, dentro da primeira chamada ao PASSO_3 (), o primeiro laço tem visibilidade de $n - 1$ vértices e o segundo laço tem visibilidade de 1 vértice. Com a realização de podas sucessivas de um único vértice, a visibilidade do primeiro laço é decrementada de uma unidade e a visibilidade do segundo laço é incrementada de uma unidade. Tal comportamento pode ser generalizado pelas expressões (6) e (7), onde (6) representa o número de passos possíveis à frente ou o que também pode ser chamado de Visibilidade à Frente (VF) e (7) o número de passos possíveis para trás ou o que também pode ser chamado de Visibilidade para Trás (VT), sendo n o número de vértices. Tais expressões podem ser reduzidas à $O(n^2)$. **A quinta dedução é:** considerando que o passo 2 do algoritmo não escalonou nenhum vértice negativo junto ao vértice final, no passo 3 o número máximo de vértices negativos deve ser $n - 3$ vértices, onde dos três subtraídos, dois são os vértices de início/fim e o terceiro é positivo. Essa configuração é atendida com um arranjo sequencial de vértices como o da Figura 5. Configurações desse tipo podem ser generalizadas através da expressão (8), composta por três elementos. O primeiro elemento é $(n - 1)$, que se refere ao número máximo de visitas que a primeira varredura através de PASSO_3 () pode fazer, considerando que haverá poda. O segundo elemento é uma progressão aritmética decrescente, em que o primeiro termo é $(n - 3)$ e o último termo é 2. E o terceiro elemento é n , que representa uma varredura final com visitação a todos os vértices. Por último, a expressão (9) toma apenas a progressão da expressão (8) e demonstra a ordem de grandeza para o esforço do algoritmo.

$$VF = \sum_{i=(n-1)}^1 i = (n - 1) + (n - 2) + \dots + 2 + 1 = O(n^2) \quad (6)$$

$$VT = \sum_{i=1}^{(n-1)} i = 1 + 2 + \dots + (n - 2) + (n - 1) = O(n^2) \quad (7)$$

$$Total \ de \ Passos = \overbrace{(n - 1)}^{1^\circ \text{ Elem.}} + \overbrace{((n - 3) + (n - 4) + \dots + 2)}^{2^\circ \text{ Elem.}} + \overbrace{n}^{3^\circ \text{ Elem.}} \quad (8)$$

$$2^\circ \text{ Elem.} = \sum_{i=(n-3)}^2 (i - 1) = \left(\frac{((n - 3) + 2)(n - 3)}{2} \right) = \left(\frac{n^2 - 4n + 3}{2} \right) = O(n^2) \quad (9)$$

Finalmente, a Figura 5 apresenta o estado original de uma árvore que incorre no pior caso do algoritmo, com o arranjo sequencial. Na árvore, os vértices destacados com fundo preto são VNSG (Vértices com Valores Negativos Suficientemente Grandes). Por definição, os vértices de início e fim têm valor zero de fluxo de caixa. Já o vértice 6 tem valor positivo de fluxo de caixa.



Figura 5: Árvore Final Estado Original.

A Figura 6 representa estados da árvore da Figura 5 com todos os eventos depois de todas as varreduras realizadas com PASSO_3 (). Na árvore da Figura 6, é possível notar que em cada varredura, apenas um vértice é podado (exceto na primeira, onde dois são podados), com a indicação da letra “X” sobre as arestas. Isso faz com que o primeiro laço do passo 3 do algoritmo tenha o menor

ritmo de diminuição até que nenhuma poda seja realizada e o algoritmo finalize com a varredura final. Na primeira varredura através de `PASSO_3()`, os seis primeiros vértices são visitados, o que representa $n - 1$, o vértice 6 é enxertado no vértice 7 e a aresta entre os vértices 4 e 5 é podada. Na segunda varredura, os quatro primeiros vértices são visitados, o que representa $n - 3$, o vértice 4 é enxertado no vértice 5 e a aresta entre os vértices 3 e 4 é podada. Na terceira varredura, os dois primeiros vértices são visitados, o vértice 2 é enxertado no vértice 3 e a aresta entre os vértices 1 e 2 é podada. Na quarta varredura, o primeiro vértice é visitado, mas como ele não tem mais nenhum sucessor neste momento, o algoritmo avalia seu predecessor, que é o vértice final (vértice 7). Com isso, a ordem de visitação dos vértices nessa varredura é 1, 7, 6, 5, 4, 3 e 2, e o algoritmo encerra o trabalho. Tal configuração caracteriza a ordem quadrática da expressão (9).

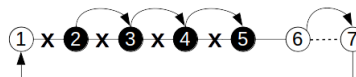


Figura 6: *Árvore Final* com todos os eventos.

7. Conclusões

A realização das três séries de avaliações cooperou para a demonstração da ordem quadrática de complexidade do terceiro passo do algoritmo de Demeulemeester et al. [1996] e sua análise expandida no trabalho de Demeulemeester e Herroelen [2002]. Essa demonstração não só confirma a robustez da solução proposta por esses autores, preenchendo uma importante lacuna teórica que esteve aberta por mais de vinte anos na literatura, bem como cria novas lacunas teóricas, na medida em que possibilita comparações mais adequadas com outras soluções propostas. Em outras palavras, dos três passos do algoritmo, os dois primeiros foram anunciados como quadráticos. No entanto, o último e mais importante dos passos se manteve sem complexidade definida, até agora. Assim, com a descoberta promovida pelas três séries deste trabalho, tem-se o caminho aberto para pesquisas futuras que envolvam comparações com outras soluções e permitam discussões de complexidade para a mesma classe do problema.

Referências

- Blazewicz, J., Lenstra, J., e Rinnooy-Kan, A. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24.
- Brucker, P. (2006). *Scheduling Algorithms*. Springer-Verlag, New York.
- Cavalcante, C., De Souza, C., Salvendy, M., Wang, Y., e Wolsey, L. (1998). Scheduling projects with labor constraints. *CORE Discussion Papers 1998059*.
- Chang, G. e Edmonds, J. (1985). The poset scheduling problem. *Order*, 2:113–118.
- Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton Univ. Press, New Jersey.
- De Souza, C. e Wolsey, L. (1997). Scheduling projects with labor constraints. Technical Report IC-97-22, Instituto de Computação - UNICAMP.
- Demeulemeester, E. L. e Herroelen, S. W. (2002). *Project Scheduling: A Research Handbook*. Kluwer's International Series, New York.
- Demeulemeester, E. L., Herroelen, S. W., e De Reyck, B. (1997). A classification scheme for project scheduling problems. Technical Report 2, Katholieke Universiteit Leuven.
- Demeulemeester, E. L., Herroelen, S. W., e Van Dommelen, P. (1996). An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem. *Katholieke Universiteit Leuven*.
- Graham, R., Lawler, E., Lenstra, J., e Rinnooy-Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of D. Math.*, ANDM 5:287–326.

- Grinold, R. C. (1972). The payment scheduling problem. *Naval R. Logistics Quart.*, 19:123–136.
- Gröflin, H., Liebling, T., e Prodon, A. (1982). Optimal subtrees and extensions. *Annals of Discrete Math.*, 16:121–127.
- Herroelen, W., Van Dommelen, P., e Demeulemeester, E. (1997). Project networks models with discounted cash flows: A guided tour through recent developments. *European Journal of Operational Research*, 100:97–121.
- Juang, S. H. (1994). Optimal solution of job-shop scheduling problems – a new network flow approach. Phd thesis, The University of Iowa, USA.
- Lacerda, I. M. e Schmitz, E. A. (2017). Valor presente líquido em projetos de software com e sem restrição de recursos. In *Anais da IV Escola Regional de Sistemas de Informação do Rio de Janeiro*, Rio de Janeiro. Sociedade Brasileira de Computação (SBC).
- Leyman, P. e Vanhoucke, M. (2016). Capital- and resource-constrained project scheduling with net present value optimization. *European Journal of Operational Research*.
- Margot, F., Prodon, A., e Liebling, T. (1990). The poset scheduling problem. *Methods of Oper. Research*, 62:221–230.
- Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, New York.
- Rodrigues, R. d. F. (2009). Caracterizações e algoritmos para problemas clássicos de escalonamento. Tese (doutorado em engenharia de sistemas e computação). universidade federal do rio de janeiro, COPPE - UFRJ.
- Roundy, R., Maxwell, W., Herer, Y., Tayur, S., e Getzler, A. (1991). A price-directed approach to real-time scheduling of production operation. *IIE Transac.*, 23:149–160.
- Russell, A. (1970). Cash flows in networks. *Management Science*, 16:357–373.
- Sankaran, J. K., Bricker, D. L., e Juang, S. H. (1999). A strong fractional cutting-plane algorithm for resource-constrained project scheduling. *International Journal of Industrial Eng.*, 6:99–111.
- Schwindt, C. e Zimmermann, J. (2001). A steepest ascent approach to maximizing the net present value of projects. *Mathematical Methods of Operations Research*, 53:435–450.
- Szwarcfiter, J. L. e Markenzon, L. (2010). *Estrutura de Dados e Seus Algoritmos*. LTC, RJ.
- Vanhoucke, M., Demeulemeester, E., e Herroelen, W. (1999). On maximizing the net present value of a project under resource constraints. Research report 9915, Department of Applied Economics, K. U. Leuven.
- Vanhoucke, M., Demeulemeester, E., e Herroelen, W. (2000). A validation of procedures for maximizing the net present value of a project: Comparison of procedures. Research report 0030, Department of Applied Economics, K. U. Leuven.