Um Avaliador Automático de Eficiência de Algoritmos para Ambientes Educacionais de Ensino de Programação

Erick Jonh F. Costa[1], Jorge Gabriel G. de S. Ramos[1], Yuri de A. Malheiros Barbosa[1], Gilberto Farias de S. Filho[2], Alisson V. Brito[2]

[1]Departamento de Ciências Exatas – Universidade Federal da Paraíba(UFPB)
Rio Tinto – PB – Brasil

[2]Centro de Informática – Universidade Federal da Paraíba(UFPB)
João Pessoa – PB – Brasil

{erick.costa, jgabriel, yuri}@dce.ufpb.br, {gilberto, alisson}@ci.ufpb.br

Abstract. *There are important characteristics that must be considered in the design of algorithms that are key to your use of them is efficiency. Therefore, it is necessary that this aspect is emphasized during the teaching process aprendiazem algorithms. This paper proposes an automatic solution for measuring the efficiency of algorithms by implementing a component that adopts mathematical methods that combine the use of experimental analysis and to assess the asymptotic complexity of algorithms. This component was developed and tested for IGED and MOCCA, educational tools for teaching algorithms and data structures. In our computational experiments we obtain an effective asymptotic analysis of seven classical algorithms.*

Resumo. *Existem características importantes que devem ser observados no projeto de algoritmos que são fundamentais no seu uso, uma delas é a eficiência. Portanto, se torna necessário que esse aspecto seja enfatizado durante o processo de ensino aprendiazem de algoritmos. Este trabalho propõe uma solução automática para a medição da eficiência dos algoritmos através da implementação de um componente que adota métodos matemáticos que combinam o uso de análise experimental e assintótica para avaliar a complexidade dos algoritmos. Este componente foi desenvolvido e testado para o IGED e MOCCA, ferramentas educacionais para ensino de algoritmos e estrutura de dados. Em nossos experimentos computacionais obtemos uma análise assintótica efetiva sobre sete algoritmos clássicos.*

1. Introdução

A análise de algoritmos compreende, em geral, duas dimensões: a corretude e a complexidade. A análise de corretude visa determinar se o algoritmo chega na solução desejada, enquanto que a análise de complexidade determina qual o custo desse algoritmo na busca da solução, principalmente o custo em termos do tempo de execução. As duas formas de análise são importantes no trabalho com algoritmos, e por isso são enfatizadas em disciplinas de ensino de algoritmos e estruturas de dados em cursos de Computação, Engenharias e outros. Partindo deste pressuposto, é necessário que esses dois aspectos sejam enfatizados durante o ensino e desenvolvimento de algoritmos. Neste sentido, o presente

trabalho propõe uma solução para a medição de eficiência de algoritmos através da definição e implementação de uma nova camada para ambientes educacionais de ensino de programação, propondo o uso combinado das abordagens empírica e assintótica para avaliação de complexidade dos algoritmos.

A ferramenta IGED (Interpretador Gráfico de Estrutura de Dados) e o MOCCA (Measurement of Complexity of an Certain Algorithm - Medição da Complexidade de um Determinado Algoritmo) foram projetadas para auxiliar no ensino de tais disciplinas; desta forma, é importante que essas ferramentas realizem automaticamente as análises de corretude e complexidade (eficiência) dos algoritmos implementados pelos alunos. Logo, as duas ferramentas adotaram o Avaliador de Eficiência proposto nesse trabalho.

Para se obter uma análise de eficiência, o trabalho propõe uma arquitetura para extrair a função custo $T(n)$ dos algoritmos e realizar sua classificação de eficiência assintótica a partir de um método matemático proposto.

Este trabalho possui a seguinte divisão, a Seção 2 apresenta os trabalhos relacionados da literatura; a Seção 3 apresenta o componente Avaliador de Eficiência, responsável pela classificação assintótica dos algoritmos. Na Seção 4 é demonstrado aplicações para o modelo de classificação através de sua implementação nas ferramentas IGED e no MOCCA, enquanto a seção 5 traz os resultados computacionais realizados em cima da implementação do IGED, por fim, a Seção 6 traz as conclusões do trabalho.

2. Revisão Bibliográfica

2.1. Análise de Eficiência de um Algoritmo

Segundo [Cormen 2002], analisar um algoritmo significa prever os recursos de que ele necessitará. Em geral, memória, largura de banda de comunicação ou a capacidade de computação são preocupações dos desenvolvedores, mas frequentemente é o tempo de computação que se deseja otimizar.

A análise da complexidade de um algoritmo pode seguir duas abordagens, a abordagem experimental que analisa o desempenho do algoritmo através da medição do seu tempo de execução em um computador real sobre diferentes conjuntos de dados de entrada. Nele é preciso implementar o algoritmo, sendo esta dependente de compilador e do hardware. A segunda é a abordagem teórica que analisa uma descrição de alto nível do algoritmo, caracterizando o tempo de execução como uma função do tamanho da entrada (n) do algoritmo. Para isso, utiliza-se um modelo de computação genérico com um único processador, o modelo RAM (Random Access Machine – Máquina de Acesso Aleatório). Nele as instruções são executadas uma após a outra, sem concorrência.

O modelo de RAM contém instruções encontradas em computadores reais, cujos tempos devem ser considerados constantes, tais como instruções aritméticas (soma, subtração, resto, piso, etc), de movimentação de dados (carregar, armazenar, copiar), de controle (desvio condicional, chamada e retorno de funções). O tempo de execução de um algoritmo será representado por uma função de custo $T(n)$ que representa a medida de tempo necessária para executar um algoritmo para um problema de tamanho n. Conforme [Ziviani 2004] et al., é importante enfatizar que $T(n)$ não representa diretamente o tempo de execução, mas o número de vezes que as operações relevantes são executada.

Para valores pequenos de *n*, qualquer algoritmo, mesmo que ineficiente, gastará pouco tempo. O que faz o programador escolher um algoritmo não é o seu desempenho sobre tamanhos de entrada pequenos, mas sim sobre tamanhos de entrada grandes. Estuda-se então o comportamento assintótico da função de complexidade $T(n)$, onde a preocupação é a maneira como o tempo de execução de um algoritmo aumenta à medida que o tamanho da entrada aumenta.

2.2. Trabalhos Relacionados

O sistema METRIC [Wegbreit 1975] foi desenvolvido com o objetivo de automatizar o processo de análise de algoritmos. As medidas analisadas pelo sistema são as complexidades de melhor, pior e médio caso, obtidas a partir de uma análise de programas escritos na linguagem Lisp. Seu processo de análise trabalha com a contabilidade empírica de passos básicos a serem processados durante sua execução, esta análise ainda infere equações diferenciais para o cálculo do tempo de execução total do algoritmo e o número de vezes que a operação básica da linguagem (CONS) é executada. O sistema METRIC é capaz de tratar apenas programas simples, introdutórios a linguagem Lisp, como os programas para obtenção de um elemento da lista, substituição de um elemento da lista, inversão dos valores da lista, união de listas, dentre outros. Não sendo uma ferramenta de propósito geral como é o caso da proposta deste trabalho.

O sistema ACE, Automatic Complexity Evaluator [Le Métayer 1988], é capaz de analisar algoritmos de propósito geral implementados em linguagem funcional. Partindo-se do programa inicial, uma função de complexidade de tempo é derivada. Esta função é transformada em uma função não recursiva equivalente, de acordo com o princípio de indução de recursão de [McCarthy 1961], usando uma biblioteca pré-definida de definições recursivas. Quando o sistema realiza a conversão das equações a expressão final de complexidade será uma composição de funções bem conhecidas, tais como funções exponenciais, logarítmicas, etc. O objetivo do ACE é a análise teórica da complexidade dos algoritmos, por isto não é considerado o custo de funções primitivas, desta forma a complexidade assintótica e o número de chamadas recursivas necessárias para a avaliação do programa possuem a mesma ordem de magnitude.

O ACME, Analisador de Complexidade Média [D. 1998], e o ANAC [Barbosa et al. 2001] são ferramentas para o cálculo da complexidade de algoritmos. Basicamente são programas de análise-síntese de códigos fontes escritos em uma linguagem procedural pré-estabelecida. Estes sistemas têm como componentes analisadores léxico e sintático, análogos a um compilador, mas sua síntese não é uma listagem de código ou conjunto de operações, mas sim o desenvolvimento de um cálculo baseado nas estruturas presentes no programa fonte, gerando como resultado uma equação de complexidade. O ANAC, diferente do ACME, possui um tratamento sobre as equações de complexidade geradas com o objetivo de simplificá-las, facilitando a análise do resultado pelo usuário. O ACME e ANAC não avaliam a complexidade de algoritmos recursivos, já que os mesmos não avaliam a execução do algoritmo, apenas a semântica de seus comandos.

[Souza 2009] apresenta o VisuAlg, um ambiente de programação que possui um compilador e interpretador para um pseudocódigo procedural e a funcionalidade de examinar a eficiência de um algoritmo, verificando quantas vezes uma determinada linha de código foi executada. A aplicação possui uma ferramenta chamada Perfil de Execução,

nela, as linhas do programa são exibidas com seu conteúdo e o número de vezes que foram executadas, não efetuando nenhum tipo de classificação de complexidade, apenas a constatação da quantidade de execução de cada comando básico do código analisado.

[Hoffmann et al. 2012] apresenta o RAML (Resource Aware ML), um ambiente de análise com uma linguagem de programação funcional que calcula automáticamente os recursos computacionais de um algoritmo em tempo de compilação. Para realizar as classificações, o RAML trabalha em cima do conceito de análise amortizado usando duas máquinas métricas, uma para avaliação da semântica e outra para medição do consumo heap. Após a análise, a saída da classificação de um algoritmo em RAML é a função de caracterização de complexidade do algoritmo e a equação que descreve a função, onde a maioria das análises são representadas pelo pior caso, inclusive para funções recursivas restritas.

Realizando uma análise comparativa de nossa proposta com as citadas acima, o intuito da pesquisa abrange muitos conceitos bem parecidos, só que abordados de maneiras diferentes. O METRIC e o RAML, trabalham com análise de complexidade, porém, só tratam problemas simples através de uma análise empírica dos passos básicos, limitando seu uso por conta de seus paradigmas de programação não adotadas pelo maioria dos programadores.

Como vimos na introdução deste trabalho, o que propomos é um modelo de classificaçõa de complexidade automática, esse modelo realiza classificações assintóticas a partir de uma função custo **T(n)**, que pode ser gerada por qualquer ambiente que contabilize passos básicos executados pelo algoritmo. Outra característica importante é a possibilidade de utilizar linguagem de alto-nível Python e JAVA, linguagens aplamente utilizadas dentro do ambito do desenvolvimento acadêmico e por grandes empresas mundialmente conhecidas.

3. Camada Avaliadora de Eficiência

Se for pedido que o aluno resolva um problema de ordenação usando um algoritmo específico como o *Couting Sort* que possui uma eficiência linear ($O(n)$) e o aluno implementar o algoritmo *Bubble Sort* com eficiência quadrática ($O(n^2)$), a ferramenta avaliadora poderá indicar uma falsa corretude na resolução do problema, já que os vetores resultantes serão idênticos ao final do processamento, pois os algoritmos tem a mesma eficácia, ou seja, termina com o vetor ordenado da mesma forma. Para evitar este resultado indesejado, propomos a Camada Avaliadora de Eficiência, responsável por analisar os algoritmos, classificá-los quanto a sua eficiência assintótica e compará-los, permitindo assim avaliar a corretude da solução do aluno de forma efetiva.

Esta camada terá acesso ao contador de passos básicos da ferramenta de programação, podendo contabilizar os comandos executados e a memória alocada para a execução de um algoritmo. A próxima seção detalha as técnicas propostas para a classificação automática de um algoritmo pelo Avaliador de Eficiência.

3.1. Arquitetura do Avaliador de Eficiência

O Avaliador de Eficiência será responsável por analisar os comandos executados pelo Interpretador de Comandos da ferramenta a que este esteja integrado e gerar relatórios so-

bre a eficiência dos algoritmos implementados pelo usuário. A interface de comunicação entre estes componentes será intermediada por uma base de dados, a Tabela de Eficiência.

A Tabela de Eficiência contém as seguintes informações a serem registradas a cada execução: *id*: identificador único do algoritmo; *n*: dimensão da entrada de dados; $T(n)$: passos básicos acumulados na execução do Interpretador de Comandos; $M(n)$: pico máximo de alocação de memória usado pelo algoritmo.

O Avaliador de Eficiência deve gerar várias instâncias aleatórias de tamanhos *n* como entrada para o algoritmo e armazenar a quantidade de comandos básicos contabilizados pelo contador de comandos da ferramenta, alimentando a Tabela de Eficiência.

A partir da Tabela de Eficiência são extraídos valores para a construção da função de custo do processamento $T(n)$ de um algoritmo, com esta função o gráfico pode ser plotado e visualmente comparado com outros algoritmos pré-cadastrados, além disso sua classificação assintótica de eficiência pode ser gerada a partir do procedimento matemático proposto a seguir.

3.2. Interpolação Polinomial de Lagrange

A interpolação Polinomial de Lagrange é capaz de identificar pontos em um conjunto de coordenadas (x, y) que não estão explícitos na função custo. Podemos definir um Polinômio Interpolador pelo teorema a seguir: dados $n + 1$ pontos distintos $x(0), x(1), ..., x(n)$ e $n + 1$ valores $y(0), y(1), ..., y(n)$, existe um e só um polinômio $P_n(X)$ de grau menor ou igual a *n* tal que $P_n(x(k)) = y(k)$. O polinômio na forma de Lagrange é dado por:

$$P_n(x) = \sum_{k=0}^{n} (f(x_k) l_k(x)), \tag{1}$$

onde $l_k(x)$ é expresso por:

$$l_k(x) = \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{2}$$

A classificação assintótica por Interpolação Polinomial de Lagrange define um valor que indicará o *coeficiente de classificação* (CC) da complexidade algorítmica. Dada uma função custo $T(n)$, o CC é um objeto matemático obtido através do cálculo da razão do valor desta função sobre sua derivada primeira. A nossa proposta é associar o CC a um esquema de classificação, gerado a partir de dois valores distintos α e β.

Estes valores não estão definidos explicitamente no intervalo de *n*, que representa as dimensões das instâncias tratadas pelos algoritmos, e devem ser escolhidos dentro deste intervalo para evitar a extrapolação polinomial no cálculo de $T(\alpha)$ e $T(\beta)$. Os valores numericos adotados neste trabalho foram empiricamente escolhidos para obter classificações mais estáveis. Um exemplo desta escolha pode ser observada na Figura 1. A escolha de α e β independem dos algoritmos analisados, pois seus valores são dependentes apenas das dimensões de *n*, logo, independem da função $T(n)$ analisada.

Os *fatores de classificação*(FC) representam as possíveis classificações assintóticas que a função $T(n)$ pode assumir. O esquema de classificação é realizada a partir de uma aproximação do *coeficiente de classificação* da função $T(n)$ comparadas com todos os *fatores de classificação* conhecidos. Desta forma, o valor que possuir a menor

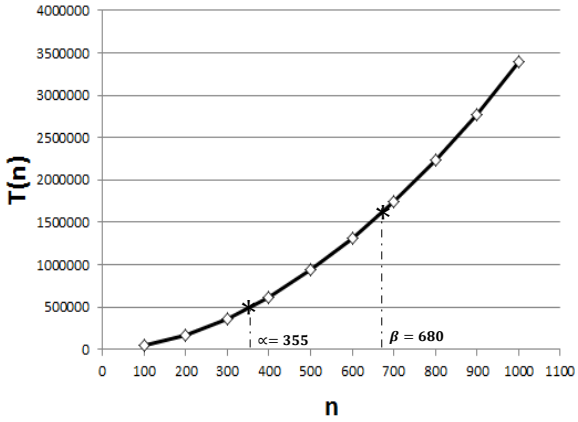

Figura 1. Valores α e β para o intervalo de dimensões das instâncias de entrada *n*.

distância irá classificar a função que caracteriza a complexidade do algoritmo.

Coeficiente de Classificação

Para identificarmos o *coeficiente de classificação* partindo dos princípios da Interpolação Polinomial de Lagrange é necessário uma equação com a função $f(n)$ e sua derivada primeira $f'(n)$ que posteriormente será comparado aos *fatores de classificação*, de modo a conseguirmos classificar a função custo do algoritmo analisado.

Para o cálculo do CC, utilizamos a equação definida em (3), para a qual atribuimos dois valores α e β como entrada:

$$CC = \frac{f(\alpha)}{f'(\alpha)} - \frac{f(\beta)}{f'(\beta)}, \tag{3}$$

onde a função $f(x)$ é dada por

$$f_n(x) = \sum_{i=0}^{n} y_i l_i(x), \tag{4}$$

Já a derivada primeira de $f(x)$ por

$$\frac{d' f_n(x)}{dx'} = \sum_{i=0}^{n} y_i \frac{d' l_i(x)}{dx'}, \tag{5}$$

onde

$$\frac{d' l_i(x)}{dx'} = \sum_{j \neq 0}^{n} \frac{1}{x_i - x_j} \prod_{k \neq i, k \neq j}^{n} \frac{x - x_j}{x_i - x_j}. \tag{6}$$

Desta forma conseguimos identificar o *coeficiente de classificação* a partir da função custo dada como entrada.

Fatores de Classificação

Para identificarmos os fatores de classificação, é necessário aplicar as expressões definidas na Tabela 2, estas expressões possuem demonstrações matemáticas que não são

apresentadas por limitações no espaço deste trabalho. A coluna *Função* define nominalmente o tipo de função atribuída à expressão, a coluna *f(x)* define a função cujo FC será definido e a coluna *FC* define as expressões que usamos para identificar o *fator de classificação* para cada função elencada e que os algoritmos classifacados podem assumir tendo os pontos α e β fornecidos como entrada.

Tabela 1. Funções $T(n)$ dos algoritmos clássicos calculados pelo componente medidor de eficiência do IGED.

Função	f(x)	FC
exponencial	ab^x	0
linear	ax	$\|\alpha - \beta\|$
quadrática	ax^2	$\|\frac{(\alpha-\beta)}{2}\|$
cúbica	ax^3	$\|\frac{(\alpha-\beta)}{3}\|$
logarítmica	$a \log_b x$	$\|\alpha ln\alpha - \beta ln\beta\|$
nlogn	$ax \log_b x$	$\|\frac{\alpha ln\alpha}{1+ln\alpha} - \frac{\beta ln\beta}{1+ln\beta}\|$
fatorial	$n!$	$\|ln\alpha - ln\beta\|$

Definindo todos esses valores, podemos classificar a função custo $T(n)$ partindo da função interpoladora de Lagrange. Após a identificação dos *fatores de classificação*, o que nos resta é comparar o *coeficiente de classificação* de $T(n)$ entre todos os *fatores de classificação* encontrados.

4. Aplicações do Modelo de Avaliação de Eficiência

Nesta seção apresentaremos a integração do Avaliador de Eficiência às ferramentas IGED e MOCCA permitindo avaliar a eficiência dos algoritmos desenvolvidos nestes ambientes.

Arquitetura Atual do IGED

O Interpretador Gráfico de Estruturas de Dados (IGED), é uma ferramenta que permite que alunos desenvolvam, numa linguagem própria, tarefas de programação passadas pelo professor, com a funcionalidade de animação das estruturas de dados utilizadas, tutoria com conteúdos hipermídia, além da checagem de corretude das estruturas finais após a execução do algoritmo do usuário. Neste trabalho é proposta uma modificação na arquitetura anterior, com a finalidade de adicionar a funcionalidade de avaliação automática de eficiência de um algoritmo ao IGED.

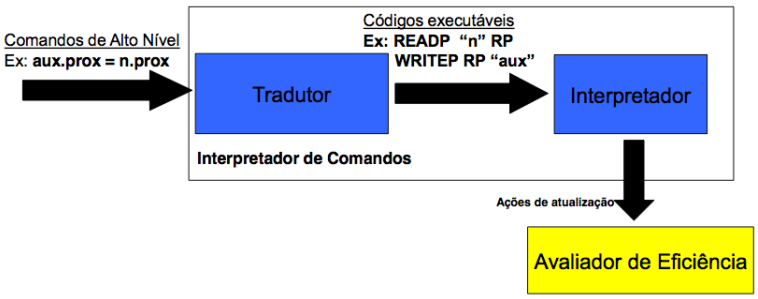

Figura 2. Fluxo de navegação do código no Interpretador de Comandos

Esta nova arquitetura é acrescida do componente Avaliador de Eficiência que alimenta sua Tabela de Eficiência através da integração com o componente Interpretador de Comandos do IGED, responsável por executar as instruções do algoritmo do usuário

e agora servindo de contador dos passos básicos gerados pela tradução do algoritmo do aluno, tarefa realizada pelo compoenente Tradutor do IGED. Esta comunicação entre os processos está ilustrada na Figura 2. Com a classificação provida pelo Avaliador de eficiência, o IGED pode comparar o resultado obtido pelos comandos passados pelo usuário com o resultado esperado em cada atividade, identificando se a solução do aluno atingiu a eficiência esperada.

Arquitetura Atual do MOCCA

O MOCCA tem o intuito de facilitar o processo de ensino aprendizagem de programação e algoritmos, sua principal funcionalidade é prover ao usuário um mecanismo de classificação, onde ele poderá verificar o desempenho de sua implementação, podendo compara-la com implementações distintas. Para isso o MOCCA implementa o avaliador de eficiência. O diferencial do MOCCA é que o mesmo trabalha com uma linguagem conhecida e amplamente utilizada por grandes empresas (Google, NASA, etc) o Python. O workflow de execução consiste basicamente em carregar um arquivo/script Python com o (os) algoritmo (os) que se deseja classificar, após carregar o arquivo, basta clicar no botão de execução e visualizar os gráficos e as classificações, havendo vários algoritmos em um mesmo arquivo, o usuário pode realizar comparações entre as diversas implementações como podemos visualizar na Figura 3.

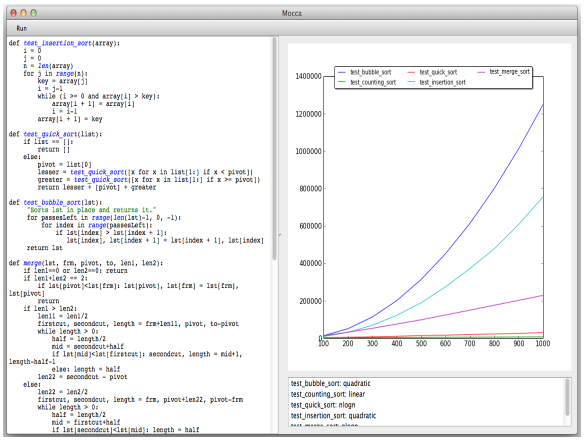

Figura 3. Captura de tela do ambiente MOCCA classificando vários algoritmos e gerando seus gráficos

O MOCCA possui um mecanismo responsável pela contagem dos passos básicos executados pelos algoritmos, sua principal funcionalidade é analisar o quanto foi gasto em termos computacionais de processamento. Ele trabalha em conjunto à Camada Avaliadora de Eficiência e executa várias instâncias diferentes para o algoritmo a fim de gerar a função T(n) que alimenta os dados da Tabela de Eficiência utilizada pelo componente para realizar a classificação automática de eficiência. A contagem dos passos básicos é feita pelo Trace, um módulo próprio da linguagem Python.

5. Resultados Computacionais

Para validarmos as metodologias de classificação de eficiência assintótica dos algoritmos implementados pelo usuário propostos no trabalho, foram realizados experimentos computacionais sobre quatro algoritmos clássicos de ordenação: *Bubble Sort*, *Counting Sort*,

Merge Sort e *Quick Sort*; um algoritmo de busca, o *Binary Search*, o algoritmo recursivo de geração dos números de *Fibonacci* e um algoritmo exaustivo para levantamento de rotas *Salesman*. Estes algoritmos foram escolhidos devido às suas diferenças na avaliação teórica de eficiência.

Os experimentos foram realizados em um computador com sistema operacional Linux Ubuntu e processador Intel Core 2 Quad 2.33Ghz com 2 GB de memória RAM.

Tabela 2. Funções $T(n)$ dos algoritmos clássicos calculados pelo componente medidor de eficiência do IGED.

n	$T(n)$					n	$T(n)$	
	bubble	**counting**	**merge**	**quick**	**binary**		**fibonnaci**	**salesman**
100	45416	3986	19020	8404	80	1	5	113
200	172680	7113	42630	19560	149	2	19	366
300	365992	10213	68010	32519	229	3	33	1622
400	611186	13313	94524	46291	309	4	61	9497
500	938493	16413	121167	58118	389	5	103	68152
600	1308529	19513	149755	75692	436	6	173	563653
700	1744667	22613	178585	93587	505	7	285	5233844
800	2234946	25713	207525	112719	585	8	467	5,373E+7
900	2775631	28813	236547	126371	665	9	761	6,042E+8
1000	3401894	31913	265180	138661	734	10	1237	7,387E+9

Os sete algoritmos foram implementados na ferramenta IGED e foram gerados 10 vetores aleatórios com diferentes dimensões crescentes, onne seus custos computacionais foram colhidos e armazenados na Tabela de Eficiência do Avaliador de Eficiência. Estes dados estão listados na Tabela 2 e representam a função custo de processamento destes algoritmos. A primeira coluna da Tabela 2 define a dimensão *n* das instâncias de vetores aleatórios testados, a coluna $T(n)$, representa a função custo de processamento de cada algoritmo avaliado, sendo esta coluna subdividida em colunas para cada algoritmo, que apresentam a quantidade de processamento de operações básicas geradas pelo Interpretador de Comandos durante a resolução de cada instância. A coluna $T(n)$ a qual pertendem os algoritmos *fibonacci* recursivo e *Salesman* exaustivo, representam os valores de *n* e $T(n)$ destes algoritmos, os valores de *n* para o algoritmo de *Fibonacci* e *Salesman* foram diferente dos demais devido a explosão exponencial de seu processamento, inviabilizando o uso dos valores originais de *n*.

5.1. Resultados Computacionais por Interpolação Polinomial de Lagrange

A Tabela 3 apresenta os *fatores de classificação* (FC) para α e β definidos por: $\alpha = 355$ e $\beta = 680$, sendo $\alpha = 2,55$ e $\beta = 7,85$ para os algoritmos *Fibonacci* e *Salesman*. A coluna *CC* indica os *coeficientes de classificação* e a coluna θ representa o limite assintótico firme de cada algoritmo cuja prova pode ser encontrada em [Cormen 2002]. Devemos observar que os *fatores de classificação* para os algoritmos *Fibonacci* e *Salesman* são os definidos a partir dos pontos $\alpha = 2,55$ e $\beta = 7,85$, logo, os *coeficiêntes de classificação de classificação* destes algoritmos deveram ser comparados a essa coluna especificamente, diferente dos demais que serão comparados aos *fatores de classificação* que foram definidos a partir dos pontos $\alpha = 355$ e $\beta = 680$.

A Tabela 4 apresenta os resultados referentes aos *coeficientes de classificação* resultantes da execução dos experimentos. A coluna Algoritmo apresenta os algoritmos estudados, a coluna CC apresenta os *coeficientes de classificação* resultantes da aplicação

Tabela 3. Valores numéricos dos *fatores de classificação* para $\alpha = 355$; $\beta = 680$ e $\alpha = 2,55$; $\beta = 7,55$.

θ	*FC*	
	$\alpha = 355; \beta = 680$	$\alpha = 2,55; \beta = 7,85$
2^n	0	0
n	325,0	5,3
n^2	162,5	2,65
n^3	108,3	1,76
$nlogn$	286,2	13,78
$logn$	2350,4	4,05
$n!$	0,64	1,12

do método de classificação para cada algoritmo e a coluna θ representa a ordem de classificação assintótica após a comparação de cada FC com os CC resultantes, definindo a ordem de classificação assintótica.

Tabela 4. Valores numéricos dos *coeficientes de classficação* para o $T(n)$ dos algoritmos analisados.

Algoritmo	**Lagrange**	
	CC	θ
bubble	160,0	n^2
counting	327,4	n
merge	290,3	$nlogn$
quick	267,3	$nlogn$
binary	2050,1	$logn$
fibonacci	0,53	2^n
salesman	1,16	$n!$

Como visto anteriormente, o melhor valor para cada algoritmo partindo do método de Lagrange, é representado pela aproximação do CC de um algoritmo com os FCs resultantes pela aplicação do método. Logo, pode-se observar que os valores resultantes do CC de cada algoritmo são exatamente, quando comparados aos FCs, os valores definidos por cada classificaçãop assintótica θ. Logo, podemos afirmar que esta técnica é eficaz na classificação automática da eficiência de um algoritmo.

6. Considerações Finais

Neste trabalho foi proposta um Avaliador de Eficiência de algoritmos para ambientes educacionais de ensino de programação. Sua função é gerar instâncias assintótcas de modo que a ferramenta contabiliza os passos básicos gerando uma função custo de processamneto, esta função será classificada pelo método matemático implementado pelo avaliador. Para validar a metodologia de classificação assintótica de eficiência dos algoritmos pelo Avaliador de Eficiência, propomos uma técnica que aborda os conceitos de Interpolação Polinomial de Lagrange sobre as funções custo de processamento dos algoritmos.

Foram realizados experimentos sobre sete algoritmos clássicos que possuem avaliações teóricas distintas. A abordagem por Lagrange obteve sucesso na classificação automática em todos os algoritmos testados, logo, podemos afirmar que para este conjunto de instâncias teste, a Interpolação Polinomial de Lagrange foi eficaz.

Diferente de outros trabalhos da literatura que apenas realizavam uma análise semántica dos algoritmos, impossibilitando por exemplo a análise de algoritmos recursivos. A camada de avaliação proposta possibilita avaliar a eficiência de forma experimentalteorica, devido a seu controle sobre o contador de passos básicos gerados pela execução

dos algoritmos. Esta abordagem foi experimentada em duas ferramentas o IGED e o MOCCA, mostrando sua eficácia na automatização da avaliação de eficiência.

Referências

Barbosa, M. A. C., Toscani, L. V., and Ribeiro, L. (2001). Anac - uma ferramenta para análise automática da complexidade de algoritmos. *Revista do CCEI*, 5(8):57–65.

Cormen, T. H. (2002). *Algoritmos: teoria e prática.* Campus, Rio de Janeiro, RJ.

D., S. C. M. (1998). Analisador de complexidade média baseado nas estruturas algorítmicas.

Hoffmann, J., Aehlig, K., and Hofmann, M. (2012). Resource Aware ML. In *24rd International Conference on Computer Aided Verification (CAV'12)*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer.

Le Métayer, D. (1988). Ace: an automatic complexity evaluator. *ACM Trans. Program. Lang. Syst.*, 10(2):248–266.

McCarthy, J. (1961). A basis for a mathematical theory of computation, preliminary report. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '61 (Western), pages 225–238, New York, NY, USA. ACM.

Souza, C. M. (2009). Visualg - ferramenta de apoio ao ensino de programação. *Revista TECCEN*, 2(2).

Wegbreit, B. (1975). Mechanical program analysis. *Commun. ACM*, 18(9):528–539.

Ziviani, N. (2004). *Projeto de algoritmos: com implementações em Pascal e C*, volume 2. Pioneira Thomson Learning, São Paulo, SP.