MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
COORDENADORIA DE INICIAÇÃO CIENTÍFICA E INTEGRAÇÃO ACADÊMICA
PROGRAMA DE INICIAÇÃO CIENTÍFICA E EM DESENVOLVIMENTO TECNOLÓGICO E INOVAÇÃO

ROGÉRIO SAMPAIO STUBS

RELATÓRIO FINAL

INICIAÇÃO CIENTÍFICA:
PIBIC CNPq ( ), PIBIC CNPq Ações Afirmativas ( ), PIBIC UFPR TN ( ), PIBIC Fundação Araucária ( ), PIBIC Voluntária (X), Jovens Talentos ( ), PIBIC EM ( ).

INICIAÇÃO EM DESENVOLVIMENTO TECNOLÓGICO E INOVAÇÃO:
PIBITI CNPq ( ), PIBITI UFPR TN ( ), PIBITI Funttel ou PIBITI Voluntária ( ).

(Período no qual esteve vinculado ao Programa 08/2016 a 07/2017)

Avaliação do Impacto do Uso de Diferentes Linguagens de Programação na Solução do Problema da Clique Máxima

Relatório Final apresentado à Coordenadoria de Iniciação Científica e Integração Acadêmica da Universidade Federal do Paraná – Edital 2016/2017.

Alexandre Prusch Züge / Campus Jandaia do Sul

Avaliação do Impacto do Uso de Diferentes Linguagens de Programação na Solução do Problema da Clique Máxima / 2016020001

JANDAIA DO SUL
2017

1. TÍTULO

Uso da Linguagem Java para a Solução do Problema da Clique Máxima

2. RESUMO

Considerando as diversas linguagens de programação disponíveis, há uma necessidade de se verificar as vantagens e desvantagens que cada uma oferece para a solução de um problema de interesse. O objetivo do projeto foi avaliar o impacto de três dessas linguagens para o Problema da Clique Máxima, que está na classe de complexidade NP-Difícil. Durante o projeto foram testados dois algoritmos que utilizam a técnica de *Branch & Bound*: Basic e MCQ. Os algoritmos foram implementados em três linguagens diferentes: Java, C++ e Python. Para viabilizar a comparação experimental das implementações, elas foram escritas de forma a gerar árvores iguais para as mesmas instâncias. Os testes foram realizados utilizando um conjunto público de instâncias e grafos aleatórios; os resultados são apresentados no final.

3. INTRODUÇÃO

O Problema da Clique Máxima consiste em encontrar o maior subgrafo completo dentro de um grafo, existem diversos algoritmos disponíveis na literatura, cada algoritmo apresenta sua vantagem e sua desvantagem, o mesmo vale para as linguagens de programação.

Existem diversas linguagens de programação, cada qual com sua vantagem e desvantagem, mas é consenso que é necessário buscar a linguagem mais adequada para cada situação. Uma linguagem de programação é um método de comunicar instruções para o computador a partir de um conjunto de regras sintáticas e semânticas. As linguagens de programação podem ser divididas em três tipos: linguagens interpretadas , linguagens compiladas e uma alternativa que combina as duas anteriores. Linguagens interpretadas, consiste na interpretação dos comandos da linguagem intermediária para a linguagem de máquina durante o tempo de execução, já as linguagens compiladas são transformadas em linguagem de máquina antes de serem executadas, enquanto as linguagens mais modernas são transformadas em uma linguagem intermediária que será interpretada pela máquina virtual da linguagem.

A complexidade computacional estuda a classificação dos problemas com base na complexidade dos algoritmos que os resolvam, a Teoria da Complexidade Computacional é dividida em diversas classes dentre elas, a classe P que consiste nos problemas que podem ser resolvidos em tempo polinomial por um algoritmo determinístico, como por exemplo o problema da Busca Linear, verificando se dado valor está em um vetor, e a classe NP consiste nos problemas que podem ser verificados em tempo polinomial, tendo como exemplo o problema do Circuito

Hamiltoniano, verificando se existe um Circuito Hamiltoniano em um grafo *G*. Os problemas de decisão (verificação) devem investigar a veracidade ou não de determinada questão, tendo como possíveis respostas sim ou não.

O problema da Clique Máxima é um problema NP-difícil Garey e Johnson(1979). Os problemas que estão em NP-Difícil, são pelo menos tão difíceis quanto os problemas mais difíceis de NP, o problema da Clique Máxima tem aplicação em diversas áreas, mas ainda não foi encontrado um algoritmo que resolva o problema em tempo polinomial para todas as suas instâncias. O problema da Clique Máxima consiste em encontrar o maior subgrafo completo no interior de um grafo.

Partindo dessas premissas o objetivo do projeto foi avaliar o impacto do uso de diferentes linguagens no tempo de processamento e na solução do problema da Clique Máxima. Foram escolhidas as linguagens Java, C++ e Python para serem testadas e analisadas. Sendo que esta parte do trabalho é referente a linguagem Java.

4. REVISÃO DA LITERATURA

Utilizamos como base para o estudo a dissertação: Solução Exata do Problema da Clique Máxima, Züge (2012), onde foi estudado complexidade computacional e algoritmos Branch & Bound e, por fim, foram definidos os algoritmos MaxCliqueBB. O problema da Clique Máxima consiste em encontrar o maior subgrafo completo no interior de outro grafo. O problema de decisão associado ao problema da Clique Máxima, isto é, dados um grafo *G* e um inteiro *k*, decidir se há clique com *k* vértices em *G*, faz parte da lista de 21 problemas clássicos NP-Completos de Karp (1972).

Os algoritmos implementados empregam a técnica Branch & Bound que consiste em um algoritmo que subdivide um problema em subproblemas menores (branching) e elimina conjuntos de subproblemas que não podem levar à solução ótima (bound), Züge (2012). A técnica utiliza a enumeração sistemática de possíveis soluções. A Fim de diminuir o número de soluções geradas (estados) são utilizados limitantes, que restringem o espaço de busca e podem melhorar o tempo de processamento do algoritmo. Em um problema de maximização como o da Clique Máxima, um limitante superior é um valor estimando que pode ser igual ou maior ao valor da solução ótima. O limitante deve ser calculado para cada estado gerado, e estados que geram cliques que só podem ser menores que a máxima não precisam ser processados.

O algoritmo MCBB (Züge 2012) é um meta-algoritmo que pode ser transformado em diversos algoritmos Branch & Bound. No algoritmo do MCBB, a variável S é uma pilha de estados, no qual as operações pop(S) e push(S), insere e remove um estado (Q, K) da pilha, onde Q e K são vetores, a variável v recebe um valor como pivô.

O primeiro algoritmo a ser implementado foi o Basic, um algoritmo bem rudimentar que a

princípio não apresenta nenhum limitante, isso o torna muito lento e impraticável, então foi adicionado um limitante trivial, fazendo com que os estados que possam gerar cliques menores da que já foi encontrada não sejam processados, economizando assim, tempo. O pseudocódigo abaixo descreve o algoritmo Basic:

```
Basic(G)
1   Entrada: Grafo G
2   Saída: Clique Máxima de G
3   (C, S) ← (Ø, pilha(Ø, V(G)))
4   Equanto S ≠ Ø
5       (Q, K) ← pop(S)
7       Enquanto K ≠ Ø & |C| < |Q| & tamanho de |K|
8           v ← recebe a última posição de K & remove a última posição de K
9           push(S,(Q,K))
10          (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v))
12      Se |C| < |Q|
13          C ← Q
14  Devolva C
```

O segundo algoritmo a ser implementado foi o MCQ, que a princípio ordena os vértices do grafo G em ordem decrescente de graus, e utiliza a coloração gulosa Greedy como limitante superior, e para elencar os vértices que serão escolhidos como pivô. O Greedy consiste em colorir um vértice de cada vez, com a menor cor possível, assim todos os vértices têm vizinhos em todas as cores menores que a sua. Intuitivamente, para um vértice pertencer a uma clique grande ele deve ter o maior número de vizinhos com cores diferentes. Diminuindo a quantidade de estados a serem processados e tornando o algoritmo mais rápido. O pseudocódigo abaixo descreve o algoritmo MCQ.

```
MCQ(G)
1   Entrada: Grafo G
2   Saída: Clique Máxima de G
3   (C, S) ← (Ø, K em ordem decrescente de graus)
4   Equanto S ≠ Ø
5       (Q, K) ← pop(S)
7       Enquanto K ≠ Ø  & |C| < |Q| & número de cores em K
8           v ←  recebe a última posição de K & remove a última posição de K
9           push(S,(Q,K))
10          (Q, K) ← (Q ∪ {v}, K ∩ Γ_G(v) )
11          Colore K com Greedy & K é colocado em ordem crescente
12      Se |C| < |Q|
13          C ← Q
14  Devolva C
```

Os algoritmos foram implementados utilizando a linguagem de programação Java, que foi desenvolvida na Sun Microsystems em 1991, por James Gosling e equipe. O grande diferencial do Java é a máquina virtual que permite executar o código fonte Java em qualquer plataforma. Em 2007 a Sun Microsystems liberou o Java como software livre, e em 2009 a Oracle adquiriu o Java. Sem dúvida Java é uma das linguagens mais utilizadas no mundo, estando disponível em 97% dos Desktops corporativos. (ORACLE, 2017)

5. MATERIAIS E MÉTODOS

Iniciamos o projeto estudando a Teoria da Complexidade Computacional, onde estudamos as classes de problemas P, NP, NP-Difícil e NP-Completos, sobre esse tema foi desenvolvido um seminário, apresentado no dia 19/10/2016, com a presença dos docentes e discentes do curso de Licenciatura em Computação.

Os algoritmos propostos em Züge (2012) utilizam a técnica Branch & Bound que baseia-se na enumeração sistemática de possíveis soluções, empregando limitantes para diminuir o número de estados gerados.

As implementações foram feitas utilizando o editor Atom[1] e executadas com a versão

[1] Ver <https://atom.io/>

1.8.0_131 da máquina virtual livre OpenJDK do Java.

Os testes foram realizados com 80 instâncias do DIMACS Second Implementation Challenges[2] e 20 mil grafos aleatórios, e foi colocado um limite de tempo de execução de duas horas por instância, pois não se conhece o tempo total de processamento de algumas instâncias.

Os grafos aleatórios são uma família, por apresentarem uma determinada propriedade específica, sendo gerados a partir de um número de vértices, nesses vértices são adicionados arestas de forma aleatória, no caso das instâncias usadas, havia 50% de chance da aresta existir.

Os testes preliminares foram processados utilizando uma máquina Intel Core i3 1.80 GHz, com 4 GB de memória, rodando a distribuição Ubuntu 16.04 do sistema operacional GNU/Linux. Os testes finais foram processados utilizando uma máquina Intel Xeon E5-1650, com seis núcleos a 3.5 GHz, e 32 GB de memória, rodando a distribuição Ubuntu 16.04 do sistema operacional GNU/Linux. A execução foi automatizada utilizando o programa GNU Parallel (TANGE, 2011).

6. RESULTADOS E DISCUSSÃO

Nas Tabelas 1 e 2 estão apresentados os resultados da implementação do MCQ, sendo a Tabela 1 referente às instâncias do DIMACS e a Tabela 2 referente a instâncias aleatórias, as tabelas estão organizadas da seguinte maneira: Na primeira coluna será apresentado o nome da instância, na segunda coluna a quantidade de árvores de estados, na terceira o tempo (T) que está em segundos e na quarta a Clique máxima (ω), quando o algoritmo não encontrar a maior clique no tempo delimitado, será informado na coluna Tempo a seguinte notação: “TimeOut”.

Tabela 1- Resultado do MCQ, utilizando o conjunto de instâncias DIMACS.

(continua)

Instâncias	Estados	T(s)	CM(ω)
brock200_1	1.229.509	8,936	21
brock200_2	12.581	0,243	12
brock200_3	35.205	0,441	15
brock200_4	162.463	1,183	17
brock400_1	548.838.951	4.484,523	27
brock400_2	834.889.585	5.741,877	29
brock400_3	219.996.767	188,702	31
brock400_4	398.558.921	2.973,619	33
brock800_1	1.361.329.332	TimeOut	-
brock800_2	1.604.291.560	TimeOut	-
brock800_3	1.590.467.461	TimeOut	-

[2] Ver <http://dimacs.rutgers.edu/Challenges/>

Tabela 1- Resultado do MCQ, utilizando o conjunto de instâncias DIMACS.
(continuação)

Instâncias	Estados	T(s)	CM(ω)
brock800_4	1.471.169.009	TimeOut	-
c-fat200-1	1.076.315.999	TimeOut	-
c-fat200-2	129.415	1,195	34
c-fat200-5	1.420.459.580	TimeOut	-
c-fat500-1	1.000.533.210	TimeOut	-
c-fat500-10	1.080.274.589	TimeOut	-
c-fat500-2	1.470.003.764	TimeOut	-
c-fat500-5	1.078.873.071	TimeOut	-
C1000.9	47	0,024	12
C125.9	97	0,041	24
C2000.5	281	0,039	58
C2000.9	501	0,088	126
C250.9	59	0,038	14
C4000.5	115	0,072	26
C500.9	315	0,082	64
DSJC1000_5	185.064.437	1.242,962	15
DSJC500_5	2.694.841	13,555	13
gen200_p0.9_44	684.941	13,898	44
gen200_p0.9_55	720.251	7,858	55
gen400_p0.9_55	998.392.721	TimeOut	-
gen400_p0.9_65	1.068.482.818	TimeOut	-
gen400_p0.9_75	957.910.404	TimeOut	-
hamming10-2	1.025	0,0486	512
hamming10-4	821.497.498	TimeOut	-
hamming6-2	65	0,08	32
hamming6-4	165	0,017	4
hamming8-2	257	0,117	128
hamming8-4	82.985	0,875	16
johnson16-2-4	646.073	0,766	8
johnson32-2-4	11.504.456.735	TimeOut	-
johnson8-2-4	73	0,006	4
johnson8-4-4	355	0,019	14
keller4	24.289	0,294	11
keller5	810.346.662	TimeOut	-
keller6	596.531.656	TimeOut	-

Tabela 1- Resultado do MCQ, utilizando o conjunto de instâncias DIMACS.

(conclusão)

Instâncias	Estados	T(s)	CM(ω)
MANN_a27	76.041	9,233	126
MANN_a45	5.703.151	5.383,095	354
MANN_a81	1.060.843	TimeOut	-
MANN_a9	143	0,02	16
p_hat1000-1	484.907	3,338	10
p_hat1000-2	541.593.432	TimeOut	-
p_hat1000-3	588.343.836	TimeOut	-
p_hat1500-1	3.376.635	26,319	12
p_hat1500-2	652.815.984	TimeOut	-
p_hat1500-3	542.013.097	TimeOut	-
p_hat300-1	3.369	0,154	8
p_hat300-2	37.815	0,516	25
p_hat300-3	8.096.465	85,181	36
p_hat500-1	27.631	0,392	9
p_hat500-2	1.888.939	23,026	36
p_hat500-3	590.707.997	TimeOut	-
p_hat700-1	75.025	0,725	11
p_hat700-2	27.735.667	45,181	44
p_hat700-3	546.873.158	TimeOut	-
san1000	567.533	20,777	15
san200_0.7_1	2.471	0,146	30
san200_0.7_2	1.347	0,058	18
san200_0.9_1	923.043	6,265	70
san200_0.9_2	1.967.061	19,616	70
san200_0.9_3	1.523.135	37,538	44
san400_0.5_1	4.153	0,242	13
san400_0.7_1	242.857	4,307	40
san400_0.7_2	100.445	2,587	30
san400_0.7_3	890.035	11,051	22
san400_0.9_1	730.138.659	1.117,319	100
sanr200_0.7	363.873	2,597	18
sanr200_0.9	91.587.754	944,926	42
sanr400_0.5	561.049	3,11	13
sanr400_0.7	200.349.861	1.242,215	21

Após analisar os resultados, podemos afirmar que a linguagem de programação Java obteve um rendimento mediano, tendo um desempenho superior que Python e inferior que C++, para uma melhor análise foi utilizado grafos aleatórios.

Tabela 2- Resultado do MCQ, utilizando o conjunto de instâncias aleatórias.

Instâncias	Estados	T(s)	CM(ω)
JMR-500-1.clq	2436169	13,846	13
JMR-500-2.clq	2447291	13,252	13
JMR-500-3.clq	2270027	12,551	13
JMR-500-4.clq	2299119	12,989	13
JMR-500-5.clq	2219609	13,371	13
JMR-500-6.clq	2569799	13,868	13
JMR-500-7.clq	2318805	13,416	13
JMR-500-8.clq	2567947	14,52	13
JMR-500-9.clq	1909137	11,242	14
JMR-500-10.clq	2438173	13,899	13
JMR-500-11.clq	2280395	12,366	13
JMR-500-12.clq	2408707	13,933	13
JMR-500-13.clq	2450713	14,622	13
JMR-500-14.clq	2450243	14,03	13
JMR-500-15.clq	2458669	13,933	13
JMR-500-16.clq	2244771	12,36	13
JMR-500-17.clq	2193125	12,673	13
JMR-500-18.clq	2551975	13,74	13
JMR-500-19.clq	2554025	13,912	13
JMR-500-20.clq	2280155	13,068	13

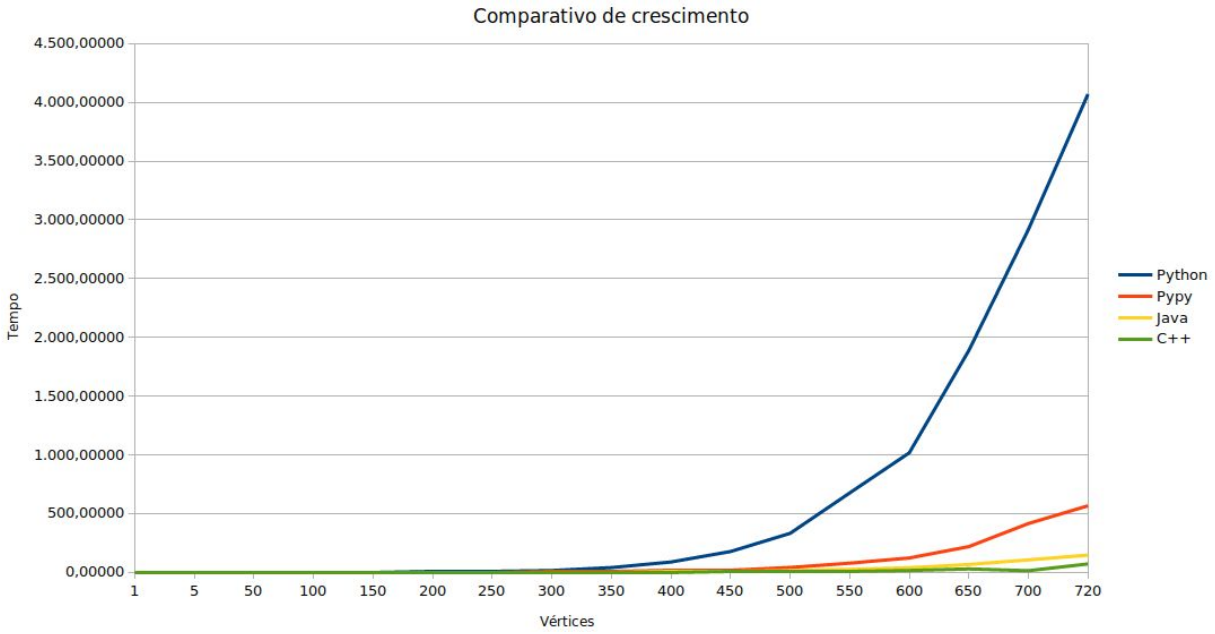

Figura 1: Comparativo entre C++, Java e Python.

Na Figura 1 fica claro a relação entre o tempo e o número de vértices, podendo observar que as linguagens Java e C++ estão muito próximas e apresentam desempenho similar.

7. REFERÊNCIAS

CORMEN, T.; LEISERSON, C.; STEIN, R. **Algoritmos: teoria e prática**. Rio de Janeiro, RJ: Elsevier Brasil, 2012.

ZÜGE, Alexandre Prusch. **Solução Exata do Problema da Clique Máxima**. 2011. 61 f. Dissertação (Mestrado) – Universidade Federal do Paraná, Curitiba, 2012.

ORACLE. **Obtendo informações sobre a tecnologia Java**. Disponível em: <https://www.java.com/pt_BR/about/>. Acessado em 18 de fevereiro de 2017.

KARP, R. M. Reducibility among combinatorial problems. In MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (Ed.). **Complexity of Computer Computations**. Springer US, 1972, p. 85-103. ISBN 978-1-4684-2003-6. Disponível em: <http://link.springer.com/chapter/10.1007%2F978-1-4684-2001-2_9.>.

TANGE, O. **GNU Parallel – the command-line power tool**. ;login: The USENIX Magazine, Frederiksberg, Dinamarca, v. 36, n. 1, p. 42–47, fev. 2011. Disponível em: <http://www.gnu.org/s/parallel>.

M.R. Garey e D.S. Johnson. **Computers and intractability**. Freeman San Francisco, 1979.

Urma, Raoul-Gabriel. **Programming language evolution**. No. UCAM-CL-TR-902. University of Cambridge, Computer Laboratory, 2017.