UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0002/2013

Estudo sobre o Tamanho dos Clusters e seus Efeitos no Cálculo do MQ

Richard Fuchshuber
Márcio de Oliveira Barros

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Relatórios Técnicos do DIA/UNIRIO, No. 0001/2013
Editor: Prof. Gleison Santos Novembro, 2013

Estudo sobre o Tamanho dos Clusters e seus Efeitos no Cálculo do MQ

Richard Fuchshuber Márcio de Oliveira Barros

Depto de Informática Aplicada – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{richard.fuchshuber, marcio.barros}@uniriotec.br

Abstract. The Modularization Quality (MQ) fitness function is often used to guide heuristic search in the resolution of software module clustering problem. This work presents a study on the effects of a restriction on the maximum and minimum sizes of each cluster. The proposed algorithm obtained worse results than the original formulation, both in terms of solution quality and runtime. It was possible to observe a tendency of MQ in yielding higher fitness values on solutions having clusters with fewer elements.

Keywords: Hill Climbing, software module clustering, mq.

Resumo. A função objetivo Modularization Quality (MQ) é muito utilizada para guiar buscas heurísticas na resolução do problema de modularização de software. Este trabalho faz um estudo sobre os efeitos da introdução de uma restrição nos tamanhos máximo e mínimo de cada cluster. O algoritmo proposto com esta restrição obteve resultados piores que o original, tanto na qualidade das soluções quanto no tempo de execução. Durante a avaliação dos resultados foi possível observar uma tendência do MQ em avaliar melhor soluções que possuam clusters com menos elementos.

Palavras-chave: Hill Climbing, modularização de software, mq.

Sumário

1 Introdução

Desenvolvedores de software frequentemente utilizam grafos direcionais para representar a estrutura complexa de um software. Tais grafos são chamados de grafos de dependência de módulos, onde os vértices do grafo representam os módulos do sistema e as arestas são os relacionamentos entre módulos [Doval et al., 1999]. Apesar de útil, a visualização desses grafos pode ser complexa até mesmo para sistemas pequenos. Uma forma de tornar a visualização mais simples é agrupar módulos relacionados em clusters, no que é chamado de problema de clusterização de software.

A criação de um grafo que represente corretamente um software é uma tarefa complexa, dada a grande quantidade de partições possíveis. Por isso, heurísticas são utilizadas para encontrar uma clusterização que melhor represente um determinado software.

O presente trabalho apresenta uma variação do algoritmo de Hill Climbing utilizado em [Barros, 2012] para gerar soluções para o problema de clusterização de software. Nesta variação, obriga-se que cada cluster tenha um número de elementos compreendido em um dado intervalo de valores, obtido através da análise do código fonte de diversos softwares. Busca-se, com isso, guiar o algoritmo de Hill Climbing de forma a encontrar soluções melhores, ao menos do ponto de vista dos desenvolvedores.

O restante do artigo está organizado como descrito a seguir. A seção dois descreve brevemente o problema de clusterização de software. Em seguida, a seção três apresenta conceitos de Hill Climbing. A seção quatro retrata a alteração proposta no algoritmo, que tem seu desempenho avaliado na seção cinco. Por fim, a seção seis trata das ameaças à validade do estudo e a seção sete apresenta as considerações finais.

2 O Problema da Clusterização de Software

A manutenção e evolução de softwares médios e grandes pode ser uma tarefa assustadora, especialmente se o sistema é mal documentado ou pior, não documentado [Doval et al., 1999]. Uma forma de entender a estrutura de um software é a criação de um grafo que o represente, conhecido como *Module Dependency Graph* (MDG) [Mitchel e Mancodiris, 2006]. No MDG, os vértices representam os módulos do sistema e as arestas os relacionamentos entre esses módulos.

O problema da clusterização de software consiste em encontrar um agrupamento dos módulos do software (clusters) que melhor represente seus conceitos de domínio e estruturas computacionais [Barros, 2012]. Uma correta distribuição de módulos facilita o desenvolvimento e manutenção do software, pois permite uma melhor compreensão do código-fonte. O problema da clusterização de software é, em sua essência, um problema de particionamento de grafos, que é conhecidamente NP-Difícil e que, por isso, pode ser tratado utilizando heurísticas.

No contexto das heurísticas, a busca da melhor representação para um software é realizada utilizando uma função objetivo como guia. Esta função avalia cada solução gerada, atribuindo-lhe um valor de forma que seja comparável a outras soluções. Para o problema da clusterização de software, as funções objetivo MQ [Doval et al., 1999] e EVM [Harman et al., 2005] são frequentemente utilizadas. Neste trabalho, o MQ foi utilizado e é introduzido a seguir.

2.1 A Função Objetivo Modularization Quality (MQ)

Modularization quality (MQ) [Mitchel e Mancodiris, 2006] é uma função objetivo baseada na conjectura de que um agrupamento de qualidade deve possuir um número baixo de relacionamentos interclusters e um alto número de relacionamentos intracluster [Doval et al., 1999]. Esta conjectura apoia-se na ideia de que um software bem desenhado é formado por um conjunto coeso de módulos que estão fracamente relacionados entre si.
MQ é composto por duas medidas. A intraconectividade mede a densidade de conexões entre os elementos de um único cluster. Uma alta intraconectividade indica um bom arranjo, pois os elementos agrupados no mesmo cluster são altamente dependentes uns dos outros. Já a interconectividade é a medida da conectividade entre os clusters. Um valor alto para a interconectividade é indesejável, pois indica que os clusters são muito dependentes uns dos outros.

A formulação do MQ [Doval et al., 1999] é apresentada na Equação I. MQ estabelece uma relação de compromisso entre interconectividade e intraconectividade, que premia a criação de clusters extremamente coesos e penaliza a criação de clusters com muitas dependências interclusters. Esta relação é alcançada pelo somatório da subtração entre a interconectividade e a intraconectividade de cada cluster.

$$MQ = \sum_{k=1}^{N} \mathrm{MF}(\mathrm{C_k}) \qquad MF(C_k) = \begin{cases} 0, & i = 0 \\ \dfrac{i}{i + j/2}, & i > 0 \end{cases} \qquad (\mathrm{I})$$

Nestas equações, N representa o número de clusters, C_k representa um cluster e i e j representam, respectivamente, o número de conexões entre clusters e o número de conexões dentro do cluster. As funções acima devem ser maximizadas, isto é, quanto maior o valor obtido, melhor é a qualidade da solução sob a perspectiva do MQ.

3 Hill Climbing

O Hill Climbing (subida de encosta) é um algoritmo de otimização iterativo, que inicia com uma solução arbitrária do problema e se move de forma contínua no sentido do valor crescente, isto é, encosta acima, alterando um elemento por vez da solução original [Russell e Norvig, 2003]. Caso uma solução melhor seja encontrada, esta nova solução é utilizada como base para a realização de outras mudanças, até que nenhuma solução de melhor qualidade seja encontrada. A Figura 1 apresenta o pseudocódigo de um Hill Climbing.

```
1.    função subida-de-encosta()
2.      no_atual ← CriarNó(ESTADO-INICIAL-ALEATÓRIO)
3.      repita
4.        vizinho ← NóVizinho(no_atual)
5.        se VALOR[vizinho] > VALOR[no_atual]
6.          no_atual ← vizinho
7.      enquanto ExisteVizinhoInexplorado(no_atual)
8.      retorna no_atual
```

Figura 1: Pseudocódigo de um Hill Climbing

A busca é guiada por uma função objetivo, que permite comparar soluções entre si e encontrar o melhor estado. O Hill Climbing frequentemente encontra soluções razoáveis para problemas cujo espaço de busca é muito grande, onde algoritmos exatos são inadequados.

Um problema do Hill Climbing é que, dada a sua característica gulosa, ele encontra apenas ótimos locais, isto é, as soluções com os maiores valores de uma vizinhança. A não ser que o espaço de busca do problema seja convexo, o algoritmo pode não atingir o máximo global. Uma forma de minimizar este problema é a combinação do Hill Climbing com reinícios aleatórios (*random-restarts*), que consiste na execução do algoritmo de Hill Climbing diversas vezes, com soluções iniciais aleatórias diferentes. A melhor solução obtida dentre todos os reinícios é retornada como a melhor solução encontrada.

Em geral, os problemas NP-Difíceis têm um número exponencial de máximos locais em que ficam paralisados. Apesar disso, um máximo local razoavelmente bom pode ser encontrado com frequência depois de um pequeno número de reinícios [Russell e Norvig, 2003].

4 Alteração Proposta no Hill Climbing

Lanza e Marinescu (2006) realizaram um estudo baseado em 45 programas desenvolvidos com a linguagem Java, onde algumas métricas são extraídas. Dentre essas métricas, os autores relatam que o número de classes em um pacote (módulo) está compreendido no intervalo fechado [6, 26], com média 17.

Este trabalho propõe uma alteração no algoritmo de Hill Climbing utilizado por [Barros, 2012] para considerar esta informação obtida em [Lanza e Marinescu, 2006]. Nesta alteração, a geração das soluções iniciais deve criar soluções cujos clusters tenham seus tamanhos compreendidos no intervalo [6, 26]. Da mesma forma, apenas soluções que satisfaçam tal restrição são consideradas durante a exploração da vizinhança de uma solução. Busca-se, com esta modificação, fornecer informações de softwares reais à heurística, no intuito de guiar melhor a busca por soluções de maior qualidade.

O tamanho máximo e mínimo permitido para uma determinada instância pode precisar ser alterado, dependendo do número de módulos existentes. Por exemplo, para uma instância com apenas 20 classes, o intervalo [6, 26] deve ser ajustado para [6, 20] já que não existem módulos suficientes na instância para originar um cluster com mais de 20 elementos.

5 Experimentos Computacionais

A fim de avaliar a eficácia da proposta apresentada na seção anterior, esta seção apresenta uma avaliação realizada entre o algoritmo de Hill Climbing utilizado por [Barros, 2012] e o Hill Climbing com a restrição no tamanho dos clusters. Os algoritmos serão comparados quanto à qualidade das soluções obtidas e quanto ao tempo de execução.

5.1 Instrumentação do Experimento

Nos experimentos deste trabalho, o Hill Climbing foi executado com *random restarts* até consumir 2000 vezes N^2 avaliações da função objetivo, onde N é o número de módulos da instância sendo executada. O experimento foi executado utilizando 12 instâncias reais, geradas a partir da análise do código-fonte de softwares *open-source* de diversos tamanhos, além de um sistema desenvolvido para uma empresa brasileira (instância *seemp*), todos desenvolvidos com a linguagem de programação Java. A Tabela 1 apresenta as características destas instâncias.

Tabela 1: Características das instâncias utilizadas no experimento

Instância	Módulos	Dependências
jodamoney	26	102
jxlsreader	27	73
seemp	31	61
javaocr	59	155
servletapi	63	131
jxlscore	83	330
jpassword96	96	361
junit	100	209
xmldom	119	276
tinytim	134	564
javacc	154	722
xmlapi	184	413

As instâncias selecionadas cobrem uma ampla gama de softwares de diferentes tamanhos, variando de pequenos sistemas com cerca de 30 módulos à sistemas médios, com cerca de 200 módulos. Neste trabalho, um módulo é um arquivo de código-fonte, possivelmente contendo mais de uma classe.

Os algoritmos foram comparados por instância, isto é, os resultados obtidos pelo Hill Climbing sem restrições quanto ao tamanho dos clusters para a instância *jodamoney* foram comparados com os resultados obtidos pelo algoritmo com a restrição, para a mesma instância. Tempos de execução menores para um determinado algoritmo indicam que ele é mais eficiente que o outro. Valores maiores para a função objetivo indicam que o algoritmo produz resultados mais efetivos que o outro.

Dado um software com *N* módulos, a busca é iniciada pela criação de *N* clusters e pela atribuição aleatória de cada módulo para um cluster. Após a distribuição inicial dos módulos nos clusters, a função objetivo MQ é calculada e o algoritmo procura por soluções melhores que a inicial, conforme ilustrado na Figura 1. Ao não encontrar soluções melhores, se o algoritmo ainda tiver processamento disponível, um reinício aleatório será realizado e a busca reinicia com uma nova solução inicial.

Os dois algoritmos foram executados 30 vezes para cada instância. Cada uma dessas execuções retornou a melhor solução encontrada, que foi armazenada junto do seu valor de avaliação dado pela função objetivo MQ. O tempo de execução também foi armazenado.

5.2 Resultados Experimentais

A Tabela 2 apresenta os resultados obtidos pelo Hill Climbing original, chamado de HC, utilizado por [Barros, 2012] (colunas 2 a 4) e os obtidos pela abordagem com a restrição no tamanho dos clusters, chamado de HC_{LIM}, (colunas 5 a 7) em relação ao valor da função objetivo MQ. A primeira coluna apresenta o nome da instância. As colunas dois, três e quatro são referentes ao Hill Climbing original e mostram, respectivamente, a melhor solução obtida entre as 30 execuções do algoritmo, a média das soluções e o desvio-padrão das 30 execuções. As colunas cinco à sete apresentam as mesmas informações para o HC_{LIM}, proposto neste artigo. Por fim, a última coluna apresenta o resultado do teste estatístico de Wilcoxon-Mann-Whitney comparando as duas versões do algoritmo. *P-values* próximos a zero indicam uma forte confiança de que os dados comparados são estatisticamente diferentes. Estes valores foram calculados utilizando o software estatístico R versão 2.15.2.

Em relação à melhor solução, o Hill Climbing original venceu a proposta HC_{LIM} em todas instâncias, obtendo soluções muito melhores. O mesmo pode ser observado em relação às médias, onde novamente todos os resultados obtidos pelo HC original superaram os resultados do HC_{LIM}. Os *P-values* não deixam dúvidas sobre a dominância do HC original, apresentando um valor menor que 0,01 para todas as instâncias do experimento.

Tabela 2: Qualidade das soluções obtidas pelo HC original e HC_{LIM}

Instância	Melhor HC	Média HC	DP HC	Melhor HC_{LIM}	Média HC_{LIM}	DP HC_{LIM}	PV
jodamoney	**2.75**	**2.74**	0.01	2.01	2.01	0	< 0.01
jxlsreader	**3.60**	**3.60**	0.01	1.89	1.89	0	< 0.01
seemp	**4.65**	**4.65**	0.01	3.27	3.27	0	< 0.01
javaocr	**9.02**	**9.00**	0.02	5.47	5.25	0.09	< 0.01
servletapi	**9.55**	**9.53**	0.02	6.20	6.12	0.05	< 0.01
jxlscore	**9.29**	**9.15**	0.08	5.80	5.59	0.11	< 0.01
jpassword96	**10.33**	**10.29**	0.02	6.66	6.40	0.11	< 0.01
junit	**11.09**	**11.09**	0	7.31	6.98	0.15	< 0.01
xmldom	**10.92**	**10.92**	0.01	7.87	7.56	0.17	< 0.01
tinytim	**12.45**	**12.38**	0.04	8.87	8.68	0.10	< 0.01
javacc	**10.54**	**10.47**	0.03	7.45	7.27	0.09	< 0.01
xmlapi	**19.02**	**18.96**	0.04	12.41	12.09	0.17	< 0.01

A Figura 2 ilustra a convergência do processo de busca ao longo do tempo. O eixo *x* mostra o tempo, representado como a fração consumida do número de utilizações da função objetivo MQ em cada posição no eixo. Como cada instância tinha N^2 avaliações disponíveis, os gráficos são apresentados em intervalos de tempo de tamanho N^2. Já o eixo vertical representa o valor da função objetivo MQ. As linhas contínuas representam a média das 30 execuções, enquanto que as linhas pontilhadas são os valores máximo e mínimo encontrados nas mesmas 30 execuções. As linhas pretas representam o HC original, enquanto que as linhas verdes representam o HC_{LIM}.

A Tabela 3 mostra os tempos médios de execução de cada instância para cada algoritmo. As colunas 2 a 5 apresentam, respectivamente, o tempo médio de execução, o desvio-padrão do tempo de execução, a média de reinícios aleatórios e o desvio-padrão para a média de reinícios aleatórios para o Hill Climbing original. As colunas 6 a 9 mostram os mesmos valores para a variante HC_{LIM} do algoritmo. Por fim, a última coluna apresenta os *p-values* ao comparar os tempos de execução de cada algoritmo.

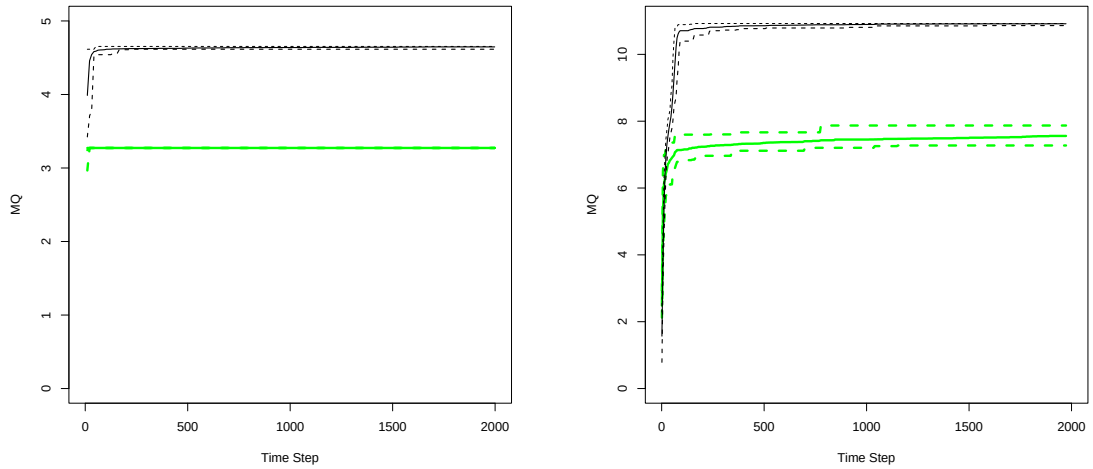

Figura 2: Evolução dos algoritmos nas instâncias SEEMP (esquerda) e XMLDOM (direita)

Tabela 3: Tempo de execução do HC original e HC_{LIM}

Instância	Média HC	DP HC	Média RA HC	DP RA HC	Média HC_{LIM}	DP HC_{LIM}	Média RA HC_{LIM}	DP RA HC_{LIM}	PV
jodamoney	**0.3**	0.02	132.5	2.1	1.3	0.03	5182.4	80.7	< 0.01
jxlsreader	**0.3**	0.01	102.6	1.6	1.3	0	3877.1	32.5	< 0.01
seemp	**0.4**	0.01	134.6	1.1	1.9	0.01	2795.6	41.7	< 0.01
javaocr	**1.9**	0.03	42.5	0.9	12.7	0.08	817.1	12.8	< 0.01
servletapi	**2.2**	0.03	45.0	1.0	15.7	0.59	702.7	11.9	< 0.01
jxlscore	**5.3**	0.14	33.9	0.9	35.5	0.32	421.8	8.2	< 0.01
jpassword96	**7.6**	0.14	22.4	0.6	58.5	1.09	339.8	7.7	< 0.01
junit	**6.9**	0.09	29.1	0.6	64.2	1.48	404.3	8.3	< 0.01
xmldom	**9.7**	0.14	26.4	0.5	105.5	1.54	347.7	8.1	< 0.01
tinytim	**18.3**	0.24	17.2	0.5	151.0	3.31	202.4	5.0	< 0.01
javacc	**25.4**	0.76	15.2	0.5	232.1	6.05	183.9	7.4	< 0.01
xmlapi	**33.8**	0.34	13.9	0.4	397.0	7.92	150.0	7.2	< 0.01

A versão original do HC apresentou tempos de execução menores em todas as instâncias. Notou-se, ainda, que conforme o tamanho da instância cresce, o esforço necessário para criar soluções que respeitem a restrição de tamanho dos clusters, bem como verificar quais vizinhos satisfazem a referida restrição, aumenta consideravelmente o tempo de execução das instâncias utilizando o algoritmo HC_{LIM}. A maior instância selecionada, *xmlapi*, teve um tempo de execução onze vezes maior utilizando o algoritmo HC_{LIM}. A restrição presente no HC_{LIM} diminuiu o número de vizinhos a serem explorados, o que permitiu um número maior de reinícios aleatórios do algoritmo, quando comparado com o HC original. A menor das instâncias, *jodamoney*, teve um número de reinícios 39 vezes maior quando o HC_{LIM} foi utilizado. Para a maior das instâncias, *xmlapi*, este valor foi quase 11 vezes maior.

A observação das soluções geradas pelo HC_{LIM} levou à suspeita de que o MQ estivesse levando a busca em direção a soluções com o maior número permitido de clusters. Para verificar tal comportamento, foram realizadas novas execuções de seis instâncias, fixando o limite inferior para o tamanho dos clusters em 4 elementos (HC_{LIM} 4) e 8 elementos (HC_{LIM} 8).

A Tabela 4 apresenta os resultados obtidos. A primeira coluna da tabela representa o nome da instância. As colunas 2 à 4 mostram, respectivamente, o número máximo de clusters permitido para cada instância, o número médio de clusters utilizados nas soluções obtidas nas 30 execuções e o valor médio do MQ dessas 30 execuções, para o Hill Climbing com a restrição de clusters com pelo menos seis elementos (HC_{LIM}). As colunas 5 à 7 mostram os mesmos resultados para a variação do algoritmo que permite clusters de no mínimo quatro elementos, chamada de HC_{LIM} 4. Por fim, as colunas 8 à 10 exibem os resultados obtidos pela variação que obriga a existência de clusters maiores, de no mínimo oito elementos (HC_{LIM} 8).

É possível observar que o MQ tende a gerar soluções com um número de clusters próximo ao máximo permitido. Ou seja, o MQ parece favorecer clusters menores. Esta constatação fica mais evidente ao se comparar o valor do MQ obtido por cada uma das variações avaliadas. O HC_{LIM} 4, que possui clusters menores, e portanto um número maior deles, supera as outras variações em todas as instâncias. Da mesma forma, o HC_{LIM} supera todos os valores do MQ ao ser comparado com o HC_{LIM} 8, que possui um número menor de clusters.

Tabela 4: Valor do MQ e número de clusters ao variar o tamanho mínimo dos clusters

Instância	HC_{LIM}			HC_{LIM} 4			HC_{LIM} 8		
	Nº Max Clusters	Nº Clusters	Média MQ	Nº Max Clusters	Nº Clusters	Média MQ	Nº Max Clusters	Nº Clusters	Média MQ
jodamoney	4	3	2.01	6	5	2.49	3	2	1.63
jxlsreader	4	3	1.89	6	5	2.54	3	2	1.49
seemp	5	4	3.27	7	6	4.25	3	2	1.91
javaocr	9	8	5.25	14	12.3	7.43	7	6	4.40
servletapi	12	9	6.12	18	13.6	8.04	9	6	4.79
jxlscore	13	11.3	5.59	20	16.4	7.05	10	8.9	4.72

6 Ameaças à Validade

As principais ameaças de conclusão em experimentos SBSE [Barros, 2012] foram endereçadas através da realização de diversas (30) execuções de cada instância para cada algoritmo, da apresentação de médias, medidas de dispersão e *p-values*. As ameaças quanto à construção do experimento foram minimizadas ao se utilizar um problema conhecido, bem como uma função objetivo (MQ) já validada pela literatura [Barros, 2012], [Doval et al., 1999], [Mitchel e Mancodiris, 2006].

Em relação as ameaças internas, este trabalho procurou utilizar os mesmos valores de parâmetros utilizados em outros trabalhos [Barros, 2012] [Barros e Farzat, 2013], além de utilizar 12 instâncias reais de tamanhos variados e de disponibilizar o código fonte em github.com/richardf/GALimitedSize. Finalmente, a utilização de instâncias de diferentes tamanhos (em relação ao número de módulos) e complexidade (em relação ao número de dependências) tratam das principais ameaças externas: a seleção das instâncias e a sua diversidade. Sobre este ponto, faz-se necessário ressaltar que embora as instâncias sejam aplicações de domínios diversos, todas foram desenvolvidas utilizando a linguagem de programação Java, de modo que os resultados apresentados podem estar restritos a softwares desenvolvidos nesta linguagem.

7 Conclusões

Este trabalho avaliou uma variação do Hill Climbing utilizado por [Barros, 2012], onde o tamanho de cada cluster deve estar compreendido no intervalo fechado de [6, 26] módulos. Este intervalo é resultado de análises realizadas por [Lanza e Marinescu, 2006] e foi incorporado ao Hill Climbing como tentativa de fornecer informações sobre sistemas reais à busca.

A solução proposta apresentou resultados estatisticamente inferiores em todas as 12 instâncias utilizadas neste trabalho, além de ter apresentado um aumento considerável no tempo de execução, decorrente do esforço computacional necessário para garantir que as soluções avaliadas respeitassem a restrição de tamanho dos clusters.

A comparação de variações do HC_{LIM} com parâmetros diferentes para o tamanho mínimo de cada cluster permitiu observar uma tendência do MQ em gerar clusters pequenos, com poucos elementos. Nesta avaliação, as soluções com o maior número de clusters, e, portanto com clusters de menor tamanho, obtiveram os maiores valores de MQ. A realização de novos experimentos, com a utilização de mais instâncias, pode trazer mais evidências sobre esta tendência observada.

Referências Bibliográficas

BARROS, M. O. (2012). **Evaluating the importance of randomness in search-based software engineering**. In: Proceedings of the 4th international conference on Search Based Software Engineering (SSBSE'12). Springer-Verlag, Berlin, Heidelberg, 60-74.

BARROS, M. O. e FARZAT, F. (2013). **What can big program teaches us about optimization?**. In: Proceedings of the 5th international conference on Search Based Software Engineering (SSBSE'13). Springer-Verlag, Berlin, Heidelberg, 60-74.

DOVAL, D., MANCODIRIS, S. e MITCHEL, B. S. (1999). **Automatic clustering of software systems using a genetic algorithm**. In: Proceedings of Software Technology and Engineering Practice.

HARMAN, M., SWIFT, S. e MAHDAVI, K. (2005). **An empirical study of the robustness of two module clustering fitness functions**. In: Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO '05), Hans-Georg Beyer.

LANZA, M. e MARINESCU, R. (2006). **Object-Oriented Metrics In Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems**. Springer-Verlag Berlin And Heidelberg Gmbh & Co. Kg.

MITCHEL, B. S. e MANCODIRIS, S. (2006). **On the automatic modularization of software systems using the bunch tool**. In: IEEE Transactions on Software Engineering, p. 193-208.

RUSSELL, J. S. e NORVIG, P. (2003). **Artificial Intelligence: A Modern Approach**, 2nd Edition, Prentice Hall.