Implementação e análise da estratégia Mutação Seletiva para redução de mutantes de primeira ordem

Alan P. C. Silva[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brazil

apcs11@inf.ufpr.br

Abstract. *This paper aims to present the Selective Mutation strategy for first ordem mutants reduction and compare it to the methods of production and reduction of high order mutants First to Last, Differente Operators, Random Mix and Each-choice.*

Resumo. *Este artigo busca apresentar a estratégia Mutação Seletiva de redução de mutantes de primeira ordem e compará-la aos métodos de produção e redução de mutantes de ordem maior First to Last, Differente Operators, Random Mix e Each-choice.*

1. Introdução

Este estudo apresentará a implementação da estratégia de redução de mutantes de primeira ordem chamada de Mutação Seletiva, assim como os resultados obtidos ao ser aplicada em 6 diferentes problemas cujos mutantes de primeira ordem já foram selecionados e a matriz de resultados dos casos de teste foram previamente calculadas.

2. Metodologia utilizada

Foram executados 6 sistemas implementados em Java já utilizados em outro trabalho de [Polo et al. 2009]. Os detalhes dos sistemas relacionados aos números de linhas de código e o número de casos de testes foram obtidos usando o plugin Source Code Metrics para Netbeans [Charkov and Warzocha 2016] e são apresentados junto ao número de FOMs para cada um na tabela (2)[1].

Tabela 1. Descrição quantitativa sobre os sistemas

Aplicação	Linhas de código	Número de casos de teste	Número de FOMs
Bisect	35	25	198
Bub	50	256	172
Find	62	135	299
Fourballs	42	96	292
Mid	57	125	317
Triangulo(TryTip)	88	216	605

Os testes usados em todos os sistemas foram implementados no seguinte trabalho [Polo et al. 2009].

[1]Os casos de teste e os sistemas utilizados neste trabalho estão acessíveis em http://bit.do/SUTs4MutationTesting.

2.1. Produção dos FOMs e casos de teste

A geração foi feita usando a ferramenta MuJava [Ma et al. 2005], a partir dos mutantes gerados foram criados casos de teste usando JUnit, uma biblioteca Java para testes. Os casos de teste gerados foram executados sobre os mutantes, resultando em uma matriz cuja combinação de linha (mutante) e coluna (caso de teste) contém 0 se o mutante não foi morto em 1 caso tenha sido morto.

Para a análise, os FOMs (mutantes de primeira ordem) vivos foram considerados como equivalente para que o *score*[2] máximo seja inicialmente 1.0, facilitando a comparação entre as estratégias.

2.2. Produção dos SOMs

As estratégias utilizadas para produção de SOMs (mutantes de segunda ordem) foram implementados por Polo[Polo et al. 2009] and Mateo[Mateo et al. 2013]. Das estratégias avaliadas foram escolhidas: *FirstToLast*, *RandomMix* and *Different-Operators*, definias respectivamente por *First2Last*, *Random* e *DiffOp*. Além dessas 3 estratégias citadas , foi escolhida mais uma de [Mateo et al. 2013]: *Each-Choice*.

Essas abordagens demostram uma redução no número de SOMs gerados mantendo a eficácia dos testes e foram escolhidas pela facilidade de produção, implementação e bons resultados obtidos.

2.3. Mutação seletiva

O objetivo da mutação seletiva é remover os FOMs gerados pelos *x* operadores que mais geraram FOMs, onde *x* é um número inteiro. Uma abordagem que parece promissora pois como os mutantes são dos operadores que mais geraram mutantes, uma quantidade significativa de mutantes será removida, que por sua vez podem ser mutantes que sejam mortos por muitos casos de uso, e caso isso se mostre verdade, serão removidos uma parcela considerável de mutantes, reduzindo o custo computacional de processamento e mantendo o *score* próximo ao original.

2.3.1. Implementação

Para implementar a estratégia de mutação seletiva foi criado um código em Ruby[3] que recebe como argumentos:

1. Diretório raíz dos problemas, por exemplo: 'sources/' (Obrigatório).
2. Conjunto com quantos *x* operadores cujos FOMs serão removidos da matriz original, por exemplo: '[2,5,7,6]' (Opcional).

Ao executar o código é criado um arquivo chamado MS_*X*.csv para cada *x* passado no segundo argumento, caso especificado. Caso não seja passado o segundo argumento, o programa irá retirar os FOMs gerados pelos 2, 5 e 10 operadores que mais geraram FOMs e inserílos em arquivos MS_*X*_removed.csv para que seja possível também analisar o complemento da estratégia seletiva.

[2]O *score* é calculado pela relação: (mutantes mortos - (total - equivalentes))/(total - equivalentes)

[3]Linguagem de programação dinâmica, mais detalhes em: https://www.ruby-lang.org/pt/

Após a aplicação da estratégia, foi executado um programa[4] que tenta encontrar o conjunto mínimo de caso de testes que mata a maioria dos mutantes, porém como este problema é complexo computacionalmete, este programa retorna um conjunto próximo do mínimo, porém não determinístico, e por esse motivo é executado 10 vezes. Os conjuntos resultantes de cada execução são escritos em arquivos com prefixo 'testcases_' para cada um dos resultados de cada estratégia.

Por fim é executado um outro código em escrito Ruby que recebe como argumentos:

1. Diretório raíz dos problemas, por exemplo: 'sources/' (Obrigatório).
2. '--summary' Flag para agrupar por média simples aritmética os scores por estratégia, e não por conjuntos de caso de teste (Opcional).

Que busca na raíz recebida como parâmetro por arquivos com o prefixo '*testcases_*'. Para cada um dos arquivos, lê os conjuntos de casos de teste únicos e para cada um calcula os scores na matriz de FOMs mortos originais obtendo como saída uma média do score para cada um dos arquivos.

Os resultados são impressos na saída padrão em JSON[5] de cada uma das estratégias comparando-os aos FOMs mortos originais, seus resultados serão apresentados a seguir.

3. Análise dos resultados

Nas seções seguintes iremos comparar resultados obtidos ao executar as estratégias HOM e Mutação Seletiva a cada problemas estudado e em seguida o *score* de cada problema por estratégia.

3.1. Análise por problemas

Primeiro iremos mostrar o desempenho de cada uma das estratégias em cada um dos seis problemas estudados.

[4]esta ferramenta foi desenvolvida utilizando um algoritmo genético de busca [Féderle, Guizzo, Colanzi, Vergilio e Spinosa, 2013]

[5]JSON é uma formatação leve de troca de dados, mais detalhes em http://www.json.org/

3.1.1. Bisect

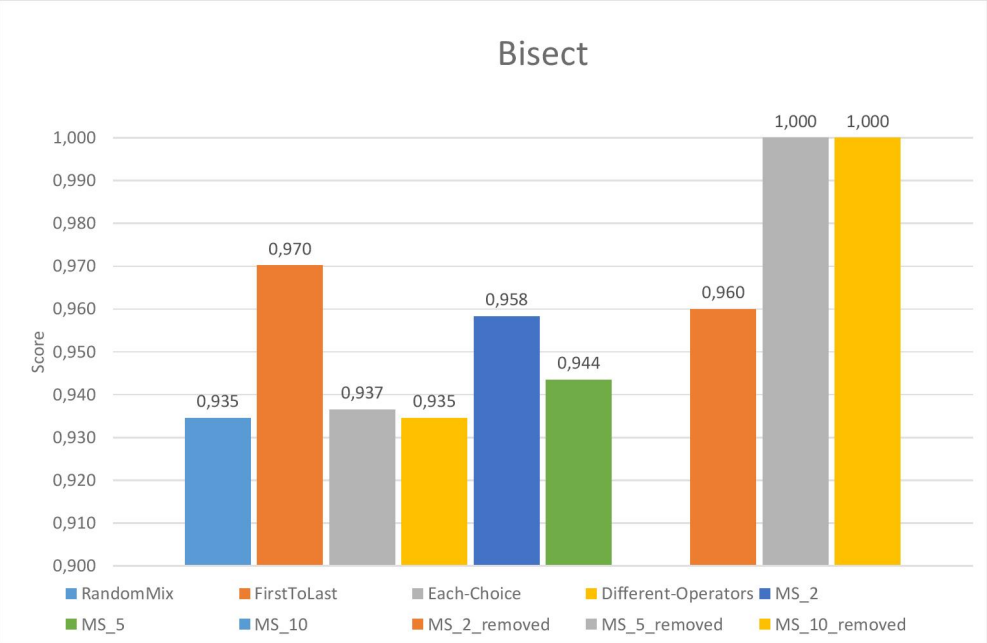

Figura 1. *Scores* das estratégias no problema Bisect

No *Bisect* podemos notar que com excessão da MS_10, as estratégias MS_2, MS_5 e MS_2_removed se saíram melhores do que 3 dos 4 SOMs. O MS_2 e seu complemento ganharam com vantagem expressiva aos 3 SOMs. A estratégia MS_10 não obteve resultado, isso aconteceu porque a matriz de FOMs fonte possuía mutantes de 10 ou menos operadores, restando nenhum mutante para ser avaliado.

3.1.2. Bub

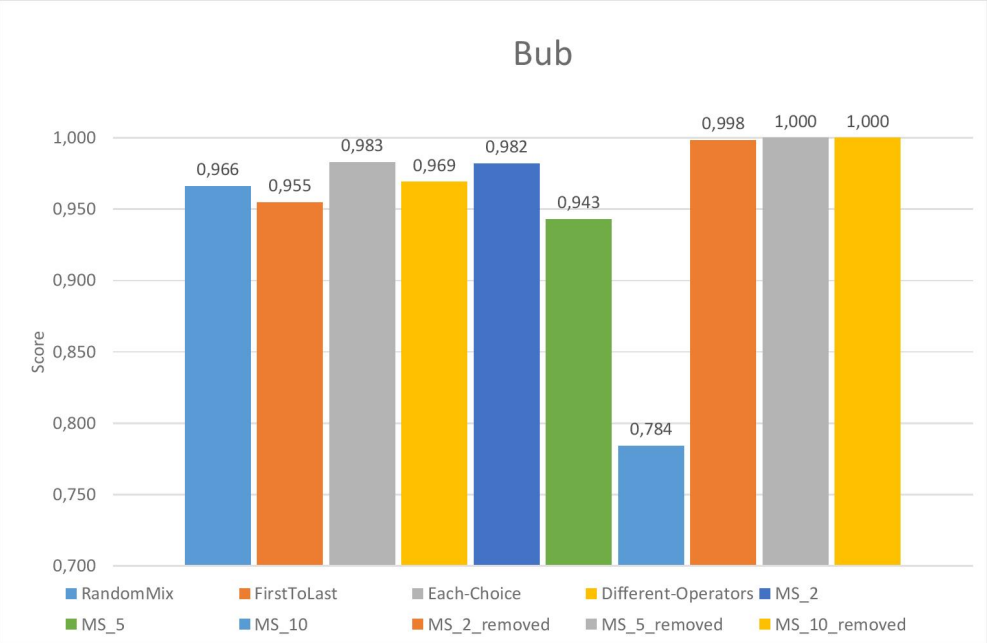

Figura 2. *Scores* das estratégias no problema Bub

No *Bub* podemos notar que novamente o MS_10 mostrou um desempenho muito abaixo da dos demais e que o MS_2 praticamente empatou com o SOM que obteve o melhor *score*. Algo a se notar nesse problema, é que os complementos ficaram quase em 1.0, e o MS_2_removed, por possui um número menor de mutantes que os demais complementos, começa parecer uma estratégia interessante de se trabalhar.

3.1.3. Find

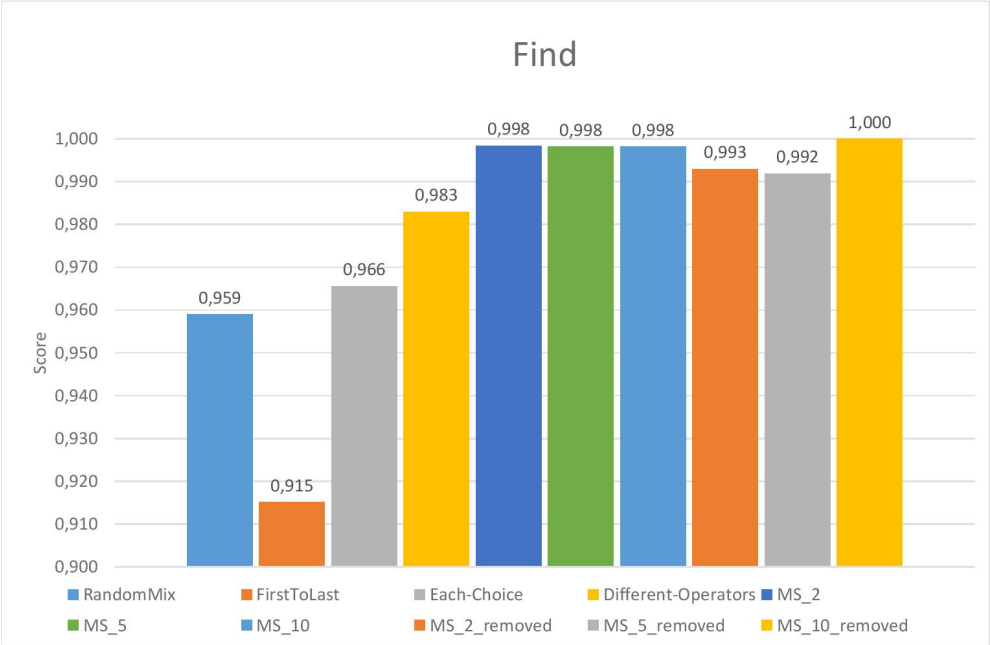

Figura 3. *Scores* das estratégias no problema Find

Neste problema, todas as estratégias de mutação seletiva se mostraram mais eficazes que os SOMs. Vale lembrar que a estratégia MS_10 tem um número muito reduzido de FOMs e que a MS_2 possui ainda menos, no entanto seus *scores* ainda ficaram próximos ao *score* ideal.

3.1.4. Fourballs

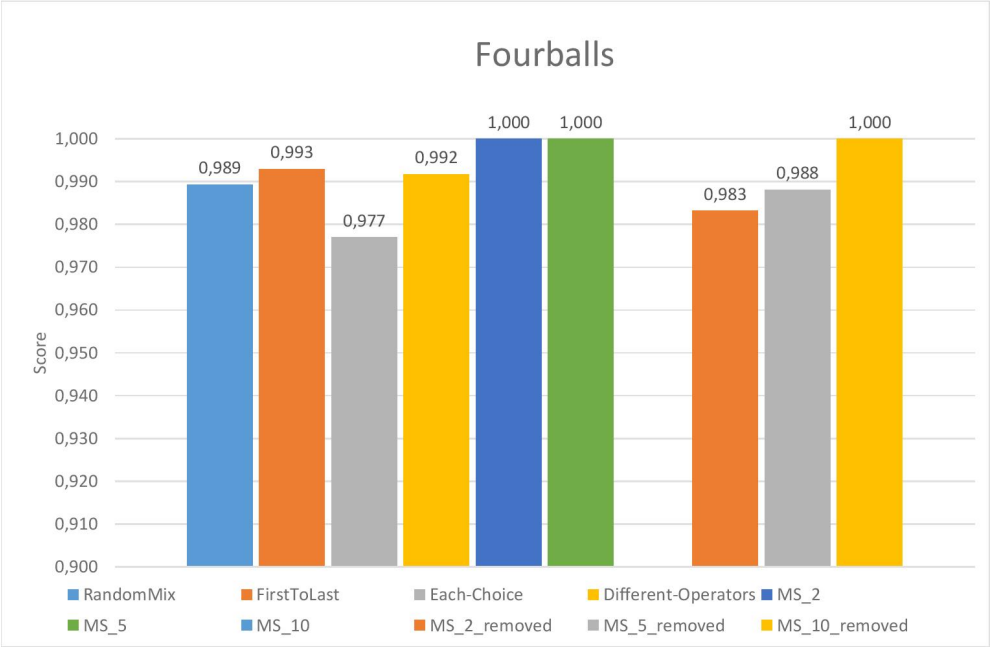

Figura 4. *Scores* das estratégias no problema Fourballs

No *Fourballs* tanto o MS_2 quanto MS_5 obtiveram *score* máximo e seus complementos ficaram próximos dos SOMs estudados, porém como nesse caso temos poucos operadores, a estratégia de mutação seletiva que terá o menor conjunto de mutantes é a MS_2. Novamente a estratégia MS_10 não obteve resultado porque a matriz de FOMs fonte possuía mutantes de 10 ou menos operadores, restando nenhum mutante para ser avaliado.

3.1.5. Mid

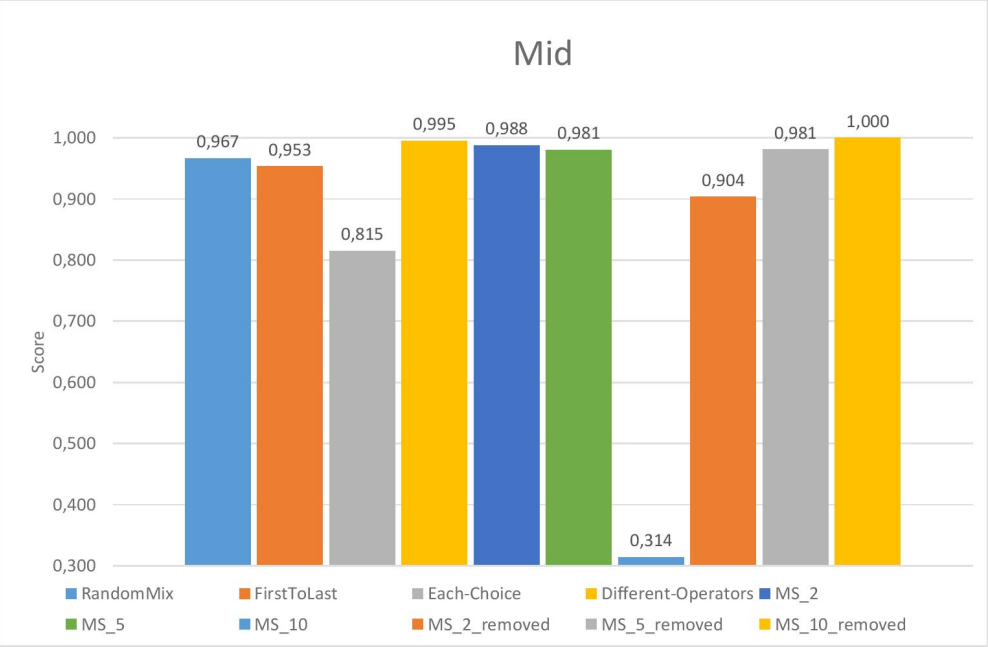

Figura 5. *Scores* das estratégias no problema Mid

A execução para este problema mostra novamente que a estratégia MS_10 possui um score muito abaixo do ideal, e que as estratégias MS_2 e MS_5 novamente se mostraram muito eficientes, perdendo por apenas aproximadamente 0,007 para o SOM *Different-Operators*.

3.1.6. Triangulo

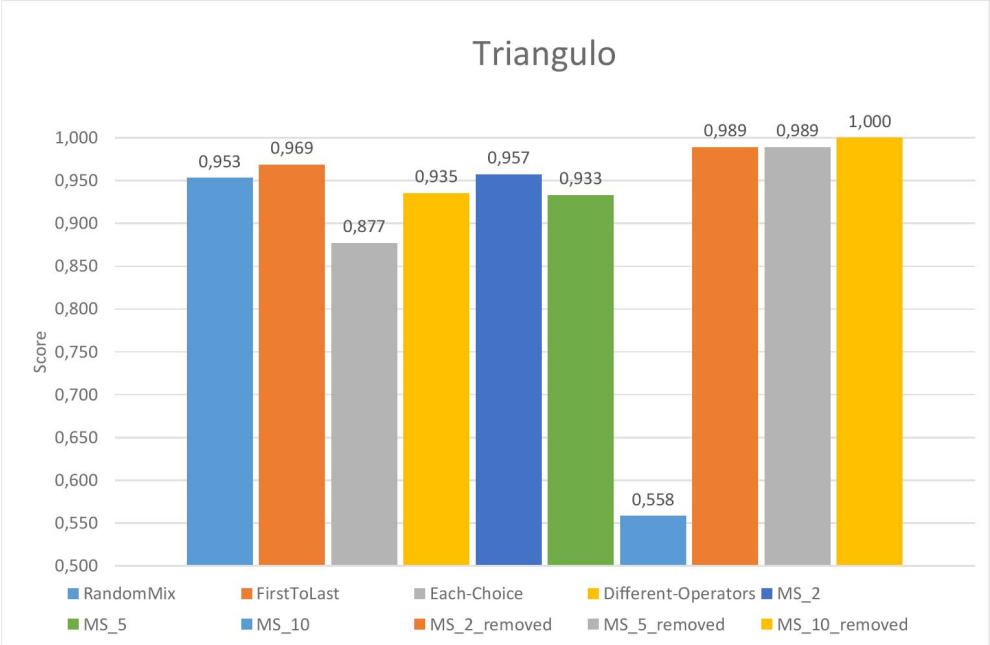

Figura 6. *Scores* das estratégias no problema Triangulo

Assim como no problema anterior, neste a estratégia MS_10 se mostra impraticável, enquanto as MS_2 e MS_5 mostram um desempenho muito próximo aos outros SOMs com melhores resultados. A MS_2 teve desempenho inferior a somente a SOM First2Last. Algo interessante neste problema é o desempenho dos complementos de MS_2 e MS_5 com *scores* próximos a 1.0.

3.2. Análise por estratégia

Nesta seção veremos o score em cada um dos problemas para cada estratégia escolhida assim como iremos fazer algumas comparações com base em alguns dados estatísticos disponíveis na tabela [2] abaixo.

Tabela 2. Dados estatísticos sobre os *scores* de cada estratégia em cada problema

	Mínimo	Máximo	Máximo-Mínimo	Média	Desvio padrão
RandomMix	0.935	0.989	0.055	0.961	0.016
FirstToLast	0.915	0.993	0.078	0.959	0.024
Each-Choice	0.815	0.983	0.168	0.926	0.061
Different-Operators	0.935	0.995	0.061	0.968	0.025
MS_2	0.957	1.000	0.043	0.981	0.017
MS_5	0.933	1.000	0.067	0.966	0.027
MS_10	0.314	0.998	0.684	0.664	0.255
MS_2_removed	0.904	0.998	0.095	0.971	0.033
MS_5_removed	0.981	1.000	0.019	0.992	0.007
MS_10_removed	1.000	1.000	0.000	1.000	0.000

3.2.1. Random

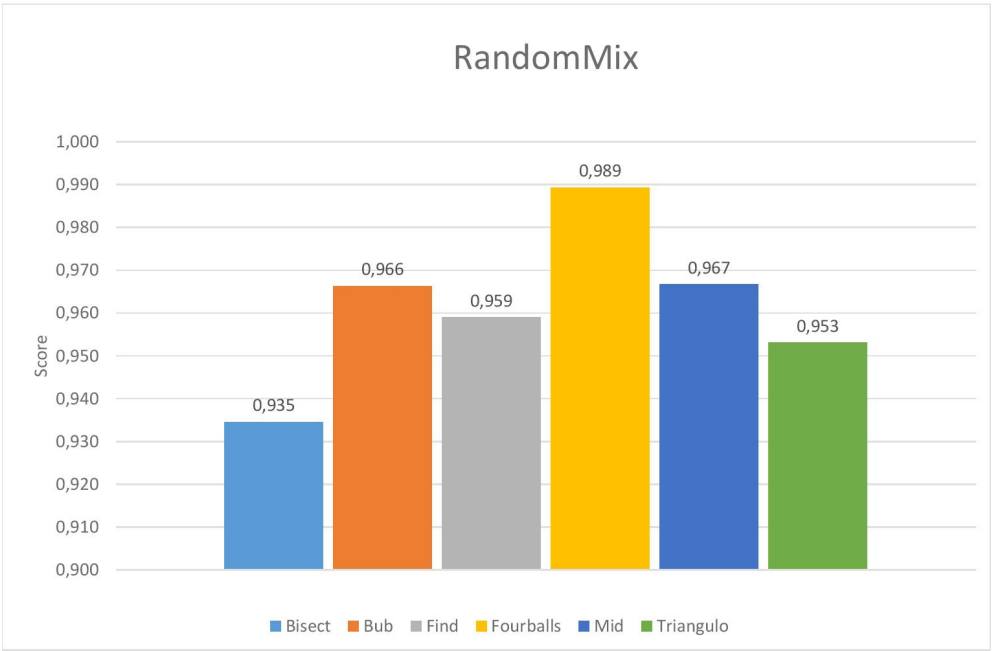

Figura 7. *Scores* dos problemas usando a estratégia Random

Com o *score* média média igual a 0.961 e desvio padrão a 0.016, podemos dizer que, no geral, esta estratégia parece ser uma solução razoável comparada às demais SOMs, porém ainda perdendo para as de Mutação seletiva.

3.2.2. First2Last

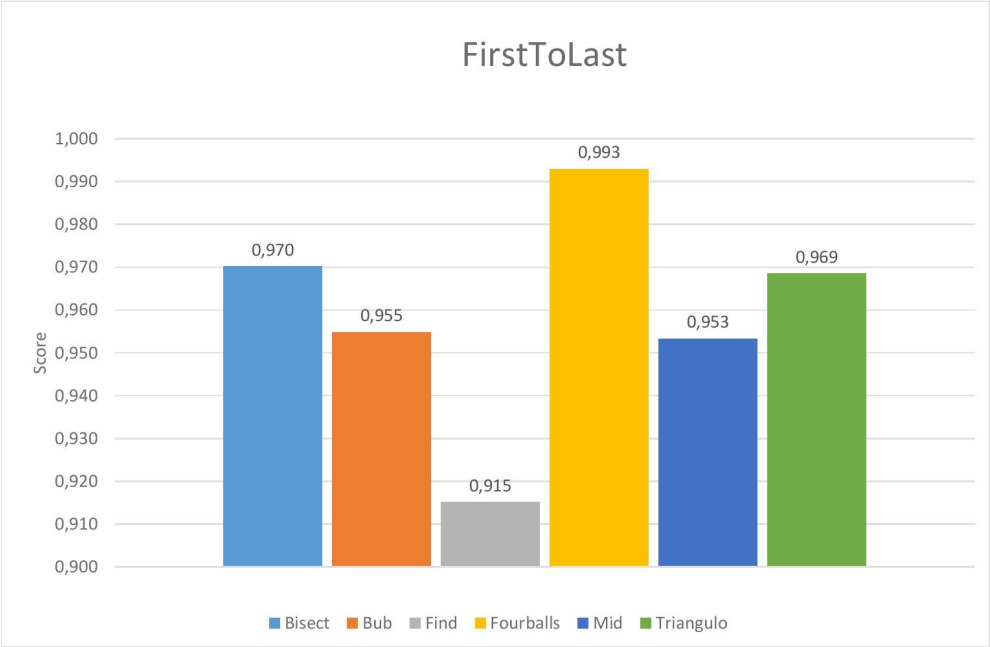

Figura 8. *Scores* dos problemas usando a estratégia First2Last

Já a *First2Last* mostra média menor (0.959) e desvio padrão maior (0.024) quando comparado à *Random*, aparentando ser uma estratégia mais inconstante, no geral.

3.2.3. Each-choice

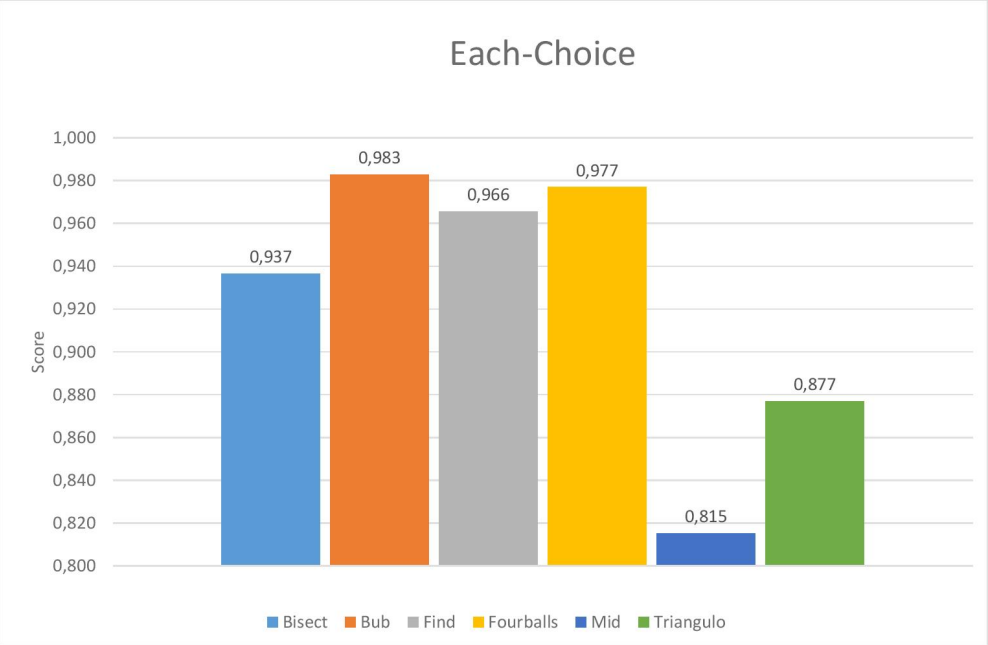

Figura 9. *Scores* dos problemas usando a estratégia Each-choice

A *Each-choice* apresenta a maior diferença entre seu máximo e mínimo dentre os SOMs (0.168), a menor média (0.926) e maior desvio padrão, se mostrando ainda mais instável do que a *First2Last*.

3.2.4. Different-Operators

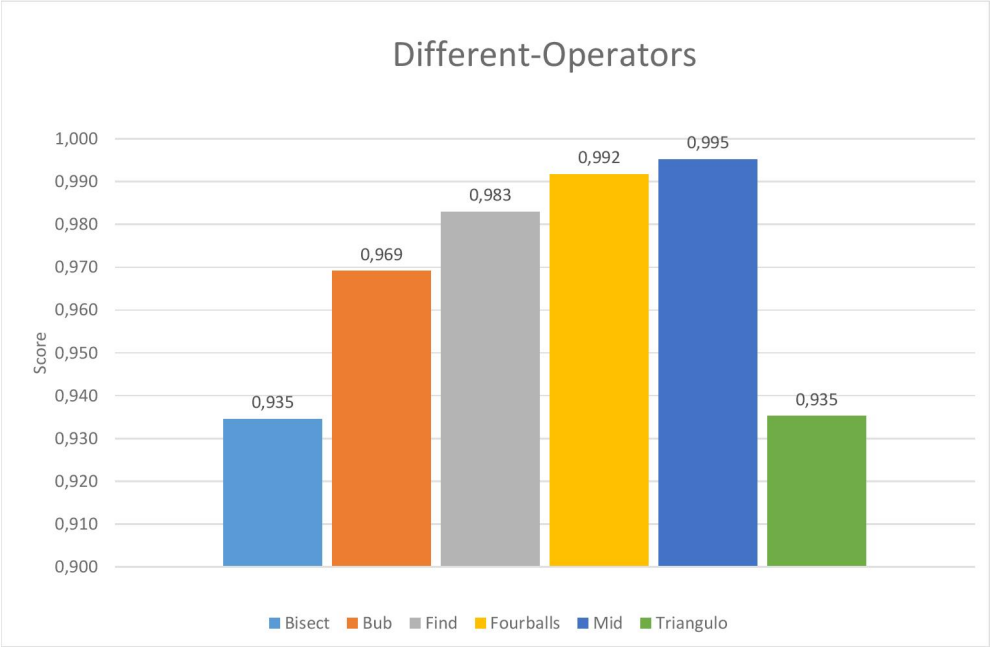

Figura 10. *Scores* dos problemas usando a estratégia Different-operators

A *Different-operators* apresentou a maior média (0.968) entre os SOMs, porém um desvio padrão elevado (0.025), no entando seus 2 maiores *scores* ficaram príxmos de 1.0.

3.2.5. Mutação seletiva

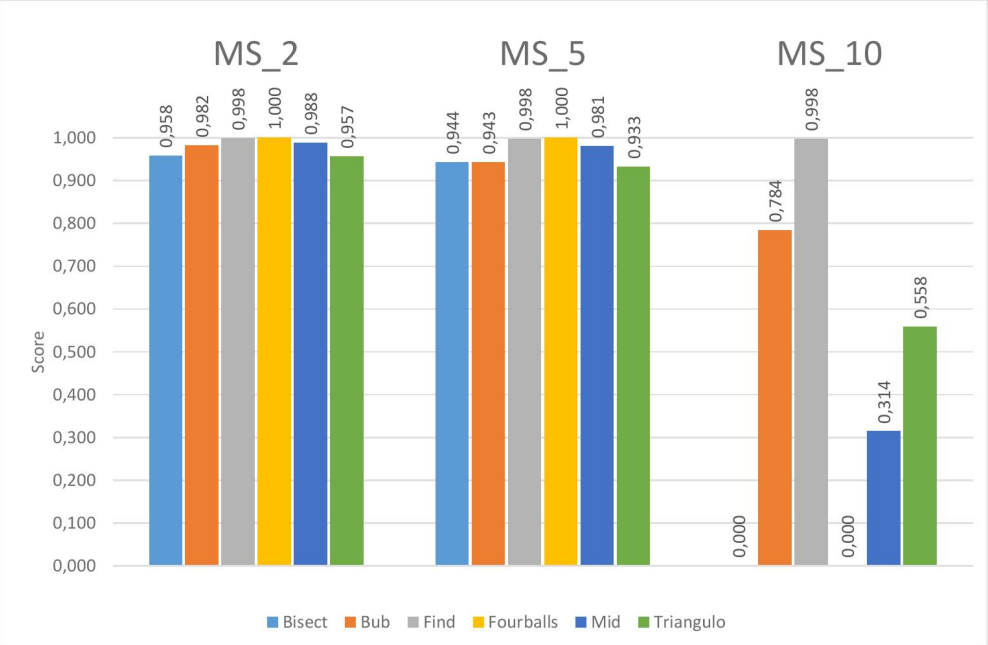

Figura 11. *Scores* dos problemas usando a estratégia Mutação seletiva

A estratégia MS_2 apresentou resultados excelentes quando comparado aos SOMs, pois seu *score* mínimo é maior que os dos SOMs e o máximo chegou a 1.0 no problema *Fourballs*. Ainda sim podemos notar que dentre as estratégias de mutação seletiva, ele foi o que se mostrou mais estável com a maior média (0.981) e com desvio padrão igual a 0.017, perdendo por uma diferença menor que 0.001 para o melhor desvio padrão dos SOMs.

A estratégia MS_5 também mostrou bons resultados, porém com maior variância entre os problemas. Assim como o MS_2, obteve um *score* máximo e outro muito próximo disso, nos mesmos problemas. Mesmo asism, sua média (0.966) ficou pior que apenas de uma estratégia SOM, a *Different-operators* e com desvio padrão (0.027) maior do que 3 das estratégias SOM.

Já a estratégia MS_10 se mostrou muito instável, pois em casos de ter poucos operadors na base inicial, restam poucos mutantes para os testes. Em dois dos problemas foram removidos todos os mutantes, porque a base inivial possuía menos de 10 operadores. Mesmo nos problemas onde sobraram mutantes para serem avaliados, seu desempenho ficou muito abaixo do esperado, com uma média de 0.664, a menor entre todas as estratégias avaliadas, e consequentemente o maior desvio padrão (0.255)

3.2.6. Mutação seletiva - Complemento

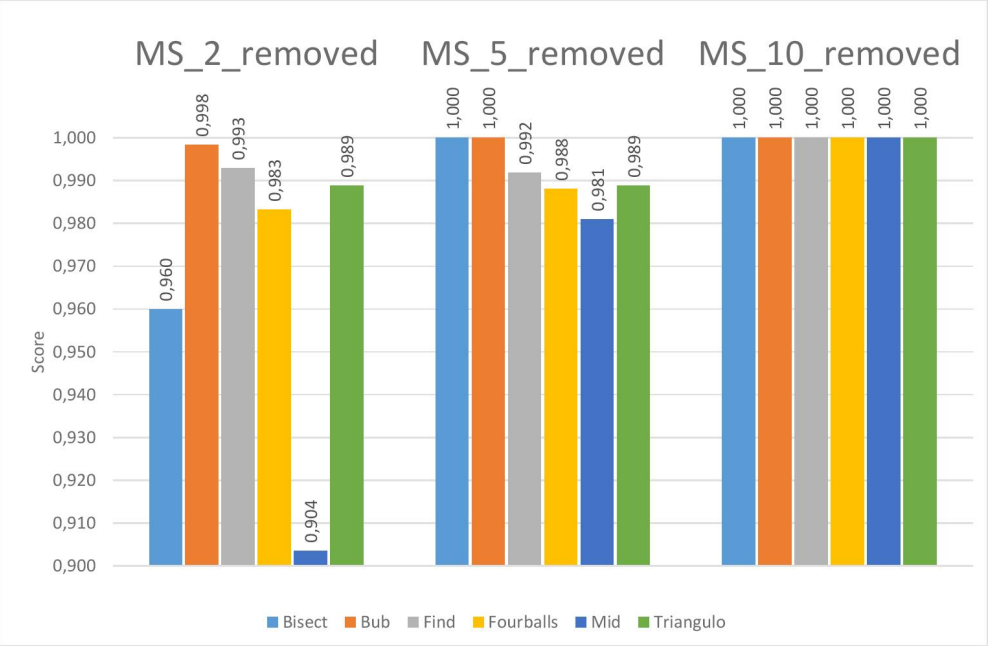

Figura 12. *Scores* dos problemas usando a estratégia Mutação seletiva - Complemento

As estratégias de mutantes removidos da mutação seletiva (complementares à ela) foram adicionados aos estudos por curiosidade e acabaram se mostrando bons resultados.

A estratégia MS_2_removed, mesmo com os mutantes de apenas 2 dos operadores das bases originais, mostrou resultados bons e próximos aos SOMs, com média de 0.971 mas com desvio padrão(0.033) elevado.

Algo surpreendente aconteceu ao ver os resultados da MS_5_removed, mesmo sendo apenas uma base complementar, ela se mostrou como a estratégia com *scores* mais estáveis dentre todos os problemas, com média 0.992 e desvio padrão igual a 0.007. Seu *score* mínimo foi de 0.981 (o maior dentre as estratégias apresentadas até agora) e o máximo 1.0.

A estratégia MS_10_removed obteve *scores* perfeitos em todos os problemas, porém em dois deles ele ficou com todos os FOMS e nos outros a maioria deles foi removida, o que acaba na verdade filtrando pouco a base inicial de FOMs, que era o proposto às estratégias.

4. Conclusão

Este estudo apresentou a implementação da estratégia de Mutação Seletiva, assim como os resultados obtidos e uma análise breve sobre seus resultados comparando-os aos mutantes

de segunda ordem propostos.

Com os resultados apresentados, foi possível ver que algumas das estratégias de mutação seletiva apresentadas foram melhores ou no mínimo empataram ficaram equiparadas às estratégias de mutantes de segunda ordem. Entretanto isso pode variar conforme o problema e quantidade de mutantes gerados. No geral, vimos que a estratégia que obteve melhor desempenho geral para os problemas propostos foi a MS_5_removed, que contém os mutantes retirados da mutação seletiva comum.

5. Considerações Finais e Trabalhos futuros

Algo que foi considerado ao fim desse estudo e análise, é que quando removemos uma quantidade muito grande de mutantes, o *score* final acaba sendo prejudicado, como ocorreu com a estratégia MS_10.

Algo que pode-se estudar para mitigar esse problema seria ao invés de remover os mutantes gerados pelos X operadores que geram mais mutantes, calcular quantos operadores foram utilizados para gerar os mutantes de primeira ordem e retirar 10%, 20% ou 50% desses operadores, por exemplo, pois assim teremos a garantia que sempre restarão mutantes para serem avaliados.

Referências

Charkov, A. and Warzocha, K. (2016). Source Code Metrics. https://github.com/alcharkov/source-code-metrics, Accessed: 2016-06-01.

Ma, Y., Offutt, J., and Kwon, Y. (2005). MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133.

Mateo, R. P., Macario, P. U., Aleman, F., and Luis, J. (2013). Validating second-order mutation at system level. *IEEE Transactions on Software Engineering*, 39(4):570–587.

Polo, M., Piattini, M., and García-Rodríguez, I. (2009). Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability*, 19(2):111–131.