



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UMA HEURÍSTICA BASEADA EM BUSCA LOCAL ITERADA PARA O
PROBLEMA DE CLUSTERIZAÇÃO DE MÓDULOS DE SOFTWARE

Alexandre Fernandes Pinto

Orientadores

Prof. Dra. Adriana Cesário de Faria Alvim

Prof. Dr. Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2014

P659 Pinto, Alexandre Fernandes.
Uma heurística baseada em busca local iterada para o problema de clusterização de módulos de software / Alexandre Fernandes Pinto, 2014.
95 f. ; 30 cm

Orientadora: Adriana Cesário de Faria Alvim.
Coorientador: Márcio de Oliveira Barros.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2014.

1. Programação heurística. 2. Iterated Local Search. 3. Clusterização de Módulos de Software. 4. Benchmarks. I. Alvim, Adriana Cesário de Faria. II. Barros, Márcio de Oliveira. III. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. IV. Título.

CDD – 005.1


UMA HEURÍSTICA BASEADA EM BUSCA LOCAL ITERADA APLICADA AO
PROBLEMA DE CLUSTERIZAÇÃO DE MÓDULOS DE SOFTWARE

Alexandre Fernandes Pinto

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA
OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-
GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO
DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO
EXAMINADORA ABAIXO ASSINADA.

Aprovada por:


Prof. Adriana Cesário de Faria Alvim, D.Sc. - UNIRIO


Prof. Márcio de Oliveira Barros, D.Sc. - UNIRIO


Prof. Alexandre Albino Andreatta, D.Sc. - UNIRIO


Prof. Leonardo Gresta Paulino Murta, D.Sc. - UFF

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2014

A minha esposa, a minha filha, a minha mãe, a meu pai (in memoriam)

Agradecimentos

Agradeço primeiramente a Deus, que me susteve e acompanhou durante este curso, permitindo ampliar os meus horizontes e a minha mente. Obrigado Pai.

Agradeço a minha família, especialmente à minha esposa Francisca Brasil, por sua compreensão e por se juntar a mim neste sonho.

Agradeço aos meus orientadores Adriana Alvim e Márcio Barros, que além da orientação exemplar e conhecimento, ofereceram sua presença constante, paciência e dedicação. Esta dedicação foi um incentivo valioso para mim.

Agradeço aos meus colegas de trabalho, em especial Wander Vasconcellos, Michel Kovacs e Fernando Pinhati, que também foram meus colegas de mestrado na Unirio. Obrigado pela troca de ideias enriquecedora que ocorreu durante este período, e pelo apoio e amizade. Agradeço também o apoio oferecido pelo TRE-RJ e por minha chefe Sonia Maria Moreira, sem o qual não seria possível cursar o mestrado.

Também sou grato aos demais colegas de mestrado, aos professores do PPGI e aos profissionais da secretaria. Obrigado pelo companheirismo, pela contribuição nas diversas disciplinas oferecidas, e pelo auxílio em momentos diversos, durante estes dois anos.

RESUMO

Este trabalho de pesquisa propõe uma heurística utilizando a técnica *Iterated Local Search* (ILS) aplicada ao problema de Clusterização de Módulos de Software (CMS). O problema CMS consiste em redistribuir os componentes unitários do software de modo a obter uma melhor organização e aprimorar a qualidade do software. A meta-heurística ILS possui quatro componentes principais: i) a solução inicial; ii) método de busca local; iii) método de perturbação e iv) critério de aceitação de uma nova solução. Um experimento foi conduzido para encontrar as melhores escolhas para os componentes de solução inicial e método de perturbação. Também foi realizado um estudo de variantes de abordagens usando Algoritmos Genéticos e um estudo comparativo entre três abordagens: a heurística utilizando ILS, Algoritmos Genéticos e um algoritmo de Busca Local. Embora a meta-heurística ILS não seja extensivamente usada na comunidade de Engenharia de Software baseada em Buscas (*Search-Based Software Engineering - SBSE*), a heurística proposta se mostrou eficaz para o problema selecionado, superando a melhor configuração do algoritmo genético em 24 das 46 instâncias, e usando uma fração do custo computacional deste último.

Palavras-chave: Iterated Local Search, Clusterização de Módulos de Software, benchmarks

ABSTRACT

This research proposes a heuristic using the *Iterated Local Search* (ILS) technique applied to the *Software Module Clustering* (SMC) problem. The SMC problem is related to the distribution of the modules comprising the software in order to obtain a better organization and improve software quality. The ILS meta-heuristic has four major components: i) the initial solution; ii) local search method; iii) perturbation method; and iv) acceptance criteria for a new solution. An experiment was conducted to find the best choices for the components related to the initial solution and perturbation method. Also we have conducted a study to compare several variants of genetic algorithms and present a comparative study including three approaches: ILS, Genetic Algorithms, and a local search. Although the ILS meta-heuristic is not extensively used in the SBSE community, it was proven very effective for our selected problem, outperforming the best configuration of the genetic algorithm in 24 out of 46 instances and using a fraction of the latter's computing effort.

Keywords: Iterated Local Search, Software Module Clustering, benchmarks

Índice

1 Capítulo 1 - Introdução	1
1.1 Definição do Problema.....	2
1.2 Motivação.....	4
1.3 Contexto de Pesquisa	5
1.4 Objetivos	6
1.5 Organização da Dissertação	7
2 Capítulo 2 - Revisão Bibliográfica	9
2.1 Busca Local e Algoritmos Genéticos Mono-objetivo.....	9
2.2 Algoritmos Genéticos Multiobjetivo.....	16
2.3 Hiper-heurísticas	18
2.4 Clusterização hierárquica	19
2.5 Algoritmos Genéticos Interativos (AGI).....	23
2.6 Incremental Search-Based-Software Engineering (ISBSE).....	25
2.7 Considerações Finais.....	26
3 Capítulo 3 - Iterated Local Search Aplicado ao Problema de Clusterização de Módulos de Software.....	28
3.1 A Meta-heurística Iterated Local Search.....	28
3.2 ILS aplicada ao problema CMS	31
3.3 Algoritmo Guloso	32
3.4 Algoritmo de Busca Local Subida de Encosta com Reinícios Aleatórios	34
3.5 Algoritmos Genéticos.....	35
3.6 Considerações finais.....	38
4 Capítulo 4 - Experimentos Computacionais	39
4.1 Instâncias de Teste	39
4.2 Parâmetros dos Experimentos Computacionais	42

4.3	Questões de Pesquisa	42
4.4	Estudo do comportamento do ILS_CMS	43
4.5	Estudo do comportamento dos algoritmos genéticos.....	50
4.6	Estudo comparativo entre diferentes abordagens.....	54
4.6.1	Comparação dos valores médios de MQ	55
4.6.2	Comparação dos tempos de execução.....	59
4.7	Estudo comparativo do ILS_CMS com instâncias de pequeno e médio porte da literatura.....	63
4.8	Características das soluções obtidas para as instâncias de testes.....	65
4.9	Ameaças à validade do estudo	68
4.10	Considerações finais.....	69
5	Capítulo 5 - Conclusões	70
5.1	Contribuições	70
5.2	Limitações e perspectivas futuras do trabalho	72
6	Referências.....	74
7	Anexo I – Configurações do ILS_CMS.....	80
8	Anexo II – <i>P-values</i> e <i>Effect-size</i> para 9 configurações do ILS	82
9	Anexo III - <i>P-values</i> e <i>Effect-size</i> GNE (one-point crossover x two-point-crossover)	83

Índice de Figuras

Figura 1. Algoritmo R-IGA (Bavota et al., 2012)	24
Figura 2. Pseudocódigo da meta-heurística ILS	29
Figura 3. Representação da busca efetuada pelo ILS	30
Figura 4. Tempo de execução dos algoritmos (em segundos) para diferentes tamanhos de instâncias (em número de módulos)	61
Figura 5. Evolução do tempo (seg.) da busca local e do ILS_CMS	62

Índice de Tabelas

Tabela 1 - Instâncias por quantidade de módulos.	40
Tabela 2 - Conjunto de 46 instâncias para testes.	40
Tabela 3. Um subconjunto das configurações avaliadas para o ILS_CMS.	44
Tabela 4. Melhor média e valor máximo do <i>MQ</i> entre as configurações do ILS_CMS	44
Tabela 5. Configurações que atingiram a melhor média em 4 ou mais instâncias.	45
Tabela 6. Quantidade de configurações que atingiram as melhores médias.	45
Tabela 7. Média e desvio padrão por instância das melhores configurações analisadas para o ILS_CMS.	46
Tabela 8. <i>P-value</i> e tamanho de efeito na comparação entre as 4 melhores configurações do ILS_CMS.	46
Tabela 9. Comparação entre as estratégias de geração da solução inicial.	48
Tabela 10. Comparativo entre as estratégias usando ou não histórico.	49
Tabela 11. Configurações avaliadas para o algoritmo genético AG_GNE.	51
Tabela 12. Configurações avaliadas para o algoritmo genético AG_GGA.	51
Tabela 13. Resultados do AG_GNE para o operador <i>one-point crossover</i>	52
Tabela 14. Resultados do AG_GNE para o operador <i>two-point crossover</i>	52
Tabela 15. <i>P-values</i> e tamanho de efeito da configuração GNE13.	53
Tabela 16. Resultados para o algoritmo genético AG_GGA.	54
Tabela 17. <i>P-values</i> e tamanho de efeito da configuração GGA6.	54
Tabela 18. Médias e desvios padrão do <i>MQ</i> para instâncias de pequeno porte.	56
Tabela 19. <i>P-value</i> e tamanho de efeito para instâncias de pequeno porte.	56
Tabela 20. Médias e desvios padrão do <i>MQ</i> para instâncias de médio porte.	57
Tabela 21. <i>P-value</i> e tamanho de efeito para instâncias de médio porte.	57
Tabela 22. Médias e desvios padrão do <i>MQ</i> para instâncias de grande porte.	58
Tabela 23. <i>P-value</i> e tamanho de efeito para instâncias de grande porte.	58
Tabela 24. Tempo de execução médio para instâncias de pequeno porte.	60
Tabela 25. Tempo de execução médio para instâncias de médio porte.	60
Tabela 26. Tempo de execução médio para instâncias de grande porte.	61
Tabela 27. Heurísticas da literatura e variantes do ILS_CMS.	64
Tabela 28. Resultados para $2000 \cdot n^2$ avaliações.	64
Tabela 29. Melhoria percentual do valor do <i>MQ</i>	65

Tabela 30. <i>P-value</i> e tamanho de efeito entre ILS_CMS* e ILS_CMS.....	65
Tabela 31. Características das melhores soluções obtidas pelo ILS_CMS.	66

Lista de Abreviaturas Utilizadas

ACDC	<i>Algorithm for Comprehension-Driven Clustering</i>
AEH	Algoritmo Evolutivo Híbrido
AG	Algoritmo Genético
AGI	Algoritmos Genéticos Interativos
AGT	Algoritmo Genético Tradicional
ATMR	<i>Attributes to methods ratio</i>
BL	<i>Busca Local</i>
BPRX	<i>Bin Packing Crossover with Replacement</i>
CF	<i>Cluster Factor</i>
CMS	Clusterização de Módulos de Software
EC	<i>External Couples Elegance</i>
ECA	<i>Equal Size Approach</i>
GGA	<i>Grouping Genetic Algorithm</i>
GNE	<i>Grouping Number Encoding</i>
HC	<i>Hill Climbing</i>
HMD	<i>Hierarchical Modular Decomposition</i>
ILS	<i>Iterated Local Search</i>
ILS_CMS	Iterated Local Search para o problema CMS
ISBSE	<i>Incremental Search-Based Software Engineering</i>
IU	<i>Internal Uses Elegance</i>
MCA	<i>Maximizing Cluster Approach</i>
MDG	<i>Module Dependence Graph</i>
MQ	<i>Modularization Quality</i>
NAC	<i>Numbers Among Class</i>
NSGA-II	<i>Non-Dominating Sorting Genetic Algorithm</i>
SBSE	<i>Search Based Software Engineering</i>
SMC	<i>Software Module Clustering</i>
SUMO	<i>Supervised Modularization</i>

Capítulo 1 - Introdução

Ao longo da sua evolução, o software sofre sucessivas modificações, tanto com manutenções corretivas quanto com o acréscimo de novas funcionalidades. Com esta evolução o software tende a aumentar em tamanho (número de componentes), em complexidade (dependências entre os componentes) e a sofrer degradação em sua estrutura interna (LEHMAN, 1979). A identificação dos componentes mais coesos funcionalmente e o ato de agrupá-los dentro das mesmas estruturas permite aumentar a compreensão e facilitar a manutenção do software. O ato de agrupar (*clusterizar*) os componentes pode, por exemplo, fornecer um modelo visual do software, identificando seus subsistemas e auxiliando desenvolvedores na compreensão de um software legado sem documentação. Outra possibilidade é aplicar a clusterização em sistemas em desenvolvimento, redistribuindo os componentes de forma automática ou semiautomática. Na redistribuição semiautomática os desenvolvedores avaliam as soluções propostas pela clusterização, refinando ou restringindo o impacto da reorganização dos componentes.

Outras vantagens oferecidas por uma boa distribuição de módulos em *clusters* são: organizar os módulos funcionalmente (LARMAN, 2001), oferecer melhor navegação entre os componentes do software (GIBBS e TSICHRITZIS, 1990) e aumentar a compreensão do software (MCCONNELL, 2004). Outro fato a ser considerado é que diversos experimentos têm demonstrado uma correlação entre a má distribuição de módulos e falhas em softwares (BRIAND et al., 1999). É possível concluir, a partir das vantagens observadas em softwares que apresentam uma boa distribuição de módulos, que a pesquisa por métodos mais eficientes e automatizados para reorganizar softwares pode contribuir para a melhoria da qualidade, redução dos gastos decorrentes de falhas e para a manutenção e evolução dos softwares.

1.1 Definição do Problema

Clusterização de Módulos de Software (CMS), em inglês *Software Module Clustering*, é o problema de distribuir os módulos componentes de um projeto de software entre estruturas maiores, chamadas de *clusters*. A distribuição se baseia nos relacionamentos existentes entre os módulos, visando colocar conjuntos de módulos que apresentam muitos relacionamentos entre si dentro de um mesmo *cluster*, enquanto módulos menos relacionados entre si são dispostos em *clusters* separados. A distribuição dos módulos em *clusters* é realizada com o objetivo de aprimorar características que indiquem a boa qualidade do software.

Os módulos considerados na organização do software podem ser, por exemplo, classes, métodos ou variáveis, enquanto os relacionamentos entre eles podem representar a importação de classes, a invocação de métodos ou o acesso de variáveis. Os *clusters* são agrupamentos de módulos, podendo representar, por exemplo, pacotes, *namespaces* ou subsistemas do software. Eles são estruturas com nível de abstração mais alto que os módulos do software. Os módulos e clusters também podem ser organizados em uma árvore de componentes aninhados, com clusters contendo outros clusters e módulos, caracterizando uma distribuição hierárquica dos componentes. Neste modelo, os módulos podem representar, por exemplo, os arquivos individuais do software, enquanto os diretórios que contém arquivos e subdiretórios podem representar os clusters. Neste trabalho é adotado somente um nível de clusterização, sem agrupamento hierárquico, e com módulos representando as classes do sistema, enquanto os clusters representam os pacotes contendo as classes.

O problema de clusterização de módulos de software não hierárquico¹ pode ser definido formalmente como a seguir (MANCORIDIS et al., 1999). Dado um conjunto N com n módulos, $N = \{M_1, M_2, M_3, \dots, M_n\}$, o problema de clusterização consiste na obtenção de um conjunto C contendo k *clusters*, tal que $C = \{C_1, C_2, \dots, C_k\}$ atenda às seguintes condições:

$$\bigcup_{i=1}^k C_i = N$$

Definição 1

¹ O problema de Clusterização de Módulos de Software pode ser estendido para uma clusterização hierárquica.

$C_i \cap C_j = \emptyset$ para $1 \leq i, j \leq k$ e $i \neq j$ Definição 2

$C_i \neq \emptyset$ para $1 \leq i \leq k$ Definição 3

A qualidade da distribuição dos módulos de um software em *clusters* é avaliada através de métricas que quantificam as características desejadas na distribuição de módulos em *clusters*. As principais características utilizadas para avaliar a qualidade de uma distribuição de módulos são acoplamento e coesão (YOURDON e CONSTANTINE, 1978). Acoplamento é uma medida de dependência entre os módulos (ou *clusters*) de um software, enquanto coesão é uma medida do quanto os integrantes do módulo (ou *cluster*) estão dedicados a um único objetivo. Usualmente, o acoplamento de um software é medido através do número de dependências entre módulos pertencentes a diferentes *clusters*, enquanto a coesão é medida através das dependências entre módulos pertencentes a um mesmo *cluster*. De modo geral, o objetivo da clusterização é distribuir os módulos do software em um determinado número de *clusters* de forma a minimizar o acoplamento e maximizar a coesão, com a restrição de que cada módulo deve pertencer a um único *cluster*.

Além de acoplamento e coesão, é possível citar outras métricas, baseadas em diferentes conceitos, para avaliar a qualidade de uma distribuição de módulos em *clusters*, tais como: *NAC (Numbers Among Classes)*, *EC (External Couple)* e *ATMR (ATtributes to Methods Ratio)* (SIMONS e PARMEE, 2012), que avaliam a simetria e elegância do software, recompensando uma distribuição mais uniforme dos módulos. Métricas de similaridade, tais como *MoJo* (TZERPOS e HOLT, 1999) e *EdgeSim* (MITCHELL e MANCORIDIS, 2007), também podem ser utilizadas na clusterização. Estas métricas permitem comparar diferentes soluções para o problema de clusterização, avaliando o quanto uma solução *A* está próxima de uma solução *B* (considerada ideal) e, deste modo, permitindo comparar a eficácia de diferentes algoritmos de clusterização.

Quanto à complexidade computacional, o problema CMS é da classe NP-Difícil (GAREY e JOHNSON, 1979), sendo essencialmente um problema de particionamento de grafos. Além disso, considerando-se que o número de *clusters* em que os módulos de um software serão distribuídos é uma variável do problema, o problema CMS representa um incremento significativo no número de soluções possíveis em relação a problemas de clusterização onde o número de *clusters* é previamente conhecido.

A estrutura dos componentes de um software, com seus módulos e relacionamentos entre eles, é usualmente representada como um grafo direcionado denominado *Module Dependence Graph* (MDG) (MANCORIDIS et al., 1999). Formalmente, o grafo $MDG = (N, R)$ consiste dos componentes N e R , onde N é o conjunto de módulos e $R \subseteq N \times N$ é o conjunto de pares ordenados $\langle u, v \rangle$ que representam as dependências entre o módulo u e o módulo v . Os grafos MDG podem ser divididos em grafos ponderados e não ponderados (com peso e sem peso nas arestas). A atribuição do peso das arestas pode ser feita de diversas maneiras. Por exemplo, a relação que representa uma chamada de função poderia valer peso 1 e a relação que representa referências às variáveis globais poderia valer peso 4, uma vez que esta última relação representa um acoplamento maior e mais prejudicial do ponto de vista das boas práticas de desenvolvimento de software. Em relação aos estudos realizados no contexto desta dissertação, estes se limitaram a tratar de MDG não ponderados.

1.2 Motivação

A literatura apresenta diversas propostas para resolução do CMS, incluindo o uso de buscas locais e algoritmos genéticos mono-objetivo e multiobjetivo (MANCORIDIS et al., 1999, MAHDAVI et al., 2003, HARMAN et al., 2005, PRADITWONG, 2011, PRADITWONG et al., 2011, BARROS, 2012, KUMARI e SRINIVAS, 2012). De acordo com o levantamento bibliográfico realizado neste trabalho, e também como observado por DOVAL et al. (1999) e HARMAN et al. (2002), nenhum algoritmo mono-objetivo superou os resultados encontrados pela busca local do tipo subida de encosta (*Hill Climbing*) com múltiplos reinícios aleatórios proposta por MANCORIDIS et al. (1998). Além disso, trabalhos recentes (PRADITWONG et al., 2011) constataram que algoritmos multiobjetivo superaram a busca local, mas seu custo de aplicação é muito superior ao custo da heurística mais simples. Tais resultados motivaram o presente estudo, que apresenta uma nova proposta de algoritmo utilizando a técnica *Iterated Local Search* (ILS) para encontrar soluções de qualidade e em tempo razoável para o problema CMS. O método proposto será comparado com a busca local do tipo subida de encosta e com algoritmos genéticos mono-objetivo.

1.3 Contexto de Pesquisa

O problema CMS se enquadra na linha de pesquisa de Engenharia de Software baseada em Buscas (*Search Based Software Engineering - SBSE*), termo cunhado por HARMAN e JONES (2001) para designar uma área de pesquisa que modela problemas da Engenharia de Software como problemas de otimização e aplica algoritmos de busca heurística para encontrar boas soluções para estes problemas (HARMAN et al., 2012).

Um problema de otimização combinatória pode ser definido (STÜTZLE, 1999) como um par (S, f) , onde S é o conjunto finito de soluções candidatas, também chamado de espaço de busca, e $f: S \rightarrow R$ é a função que atribui a cada valor $s \in S$ um valor $f(s)$, o valor da função objetivo. Sendo o objetivo do problema de otimização combinatória achar a solução $s \in S$ com o menor valor da função objetivo², chama-se de solução ótima global de (S, f) a solução s_{opt} tal que $f(s_{opt}) \leq f(s') \forall s' \in S$. O conjunto S_{opt} é o conjunto de todas as soluções ótimas globais.

Algoritmos para resolver problemas de otimização podem ser exatos ou heurísticos. Os algoritmos exatos garantem a solução ótima. Entretanto, não existem algoritmos em tempo polinomial para resolver problemas da classe NP-Difícil, admitindo-se que $P \neq NP$. Em muitos casos, os algoritmos exatos consomem extenso tempo computacional, o que os torna, com frequência, inviáveis para problemas do tipo NP-Difícil, como é o caso do problema CMS.

Por outro lado, os algoritmos heurísticos, buscam retornar boas soluções em tempo polinomial, não garantindo a solução ótima global. A palavra heurística deriva da palavra grega *heuriskein* (εὕρισκειν) que significa “achar, descobrir”. Segundo BLUM e ROLI (2003) é possível distinguir, de modo geral, duas abordagens heurísticas: construtiva e busca local. A abordagem construtiva parte de uma solução inicial parcial e vai adicionando componentes na solução de forma iterativa até obter uma solução completa viável. Na busca local, a solução corrente é substituída em cada iteração pela melhor solução em uma vizinhança previamente definida. A solução encontrada por um algoritmo de busca local é um ótimo local, na maioria das vezes diferente do ótimo global.

² O objetivo também pode ser encontrar o maior valor da função objetivo. Maximizar uma função objetivo f é o mesmo que minimizar uma função $-f$

Com o objetivo de escapar de ótimos locais, os métodos heurísticos mais básicos foram combinados, originando as meta-heurísticas³ (o sufixo meta significa “acima, em nível superior”). BLUM e ROLI (2003) apresentam diversas definições para meta-heurísticas, conforme os trabalhos de OSMAN e LAPORTE (1996), VOSS et al. (1999) e STÜTZLE (1999). Resumidamente, meta-heurísticas podem ser definidas como estratégias de nível mais alto para exploração do espaço de busca, através do uso de diferentes métodos (BLUM e ROLI, 2003). Combinam métodos heurísticos diversos, como busca local e algoritmos construtivos, com outras ideias mais sofisticadas, com o objetivo de melhorar a qualidade e eficiência. Entre essas ideias é possível citar a diversificação e intensificação durante as iterações da busca local, técnicas de múltiplos reinícios e randomização. O uso destas estratégias permite que as meta-heurísticas escapem de ótimos locais que aprisionam as heurísticas mais simples. São exemplos de meta-heurísticas *Iterated Local Search* (ILS) (LOURENÇO et al., 2001), *Simulated Annealing* (KIRKPATRICK et al., 1983), Busca Tabu (*Tabu Search*) (GLOVER e LAGUNA, 1987), Algoritmos Genéticos (HOLLAND, 1992), entre outros. O presente trabalho realiza um estudo comparativo de diferentes abordagens heurísticas aplicadas ao problema CMS, avaliando os resultados obtidos e as estratégias em cada abordagem.

1.4 Objetivos

Os objetivos desta pesquisa são:

- (i) Propor o uso de uma heurística baseada na meta-heurística *Iterated Local Search* (ILS) (LOURENÇO et al., 2001) para o CMS;
- (ii) Projetar, executar e analisar resultados de um estudo experimental de variantes de abordagens usando algoritmos genéticos para o CMS; e
- (iii) Comparar três abordagens – a heurística proposta baseada em ILS, duas heurísticas usando algoritmos genéticos e uma busca local com múltiplos reinícios aleatórios conforme apresentada em (MANCORIDIS et al., 1998) – com relação as respectivas capacidades de encontrar boas soluções para o CMS.

³ <http://www.metaheuristics.net>

Em diversos estudos comparativos (DOVAL et al., 1999, HARMAN et al., 2002) (MITCHELL e MANCORIDIS, 2002), a busca local do tipo subida de encosta com reinícios aleatórios têm obtido soluções de melhor qualidade que outras heurísticas, tais como algoritmos genéticos mono-objetivo. Em estudos recentes, algoritmos genéticos multiobjetivo produziram soluções de melhor qualidade que a busca local, porém com custo computacional muito superior. A proposta de utilização da meta-heurística ILS para resolver heurísticamente o CMS, primeiro objetivo desta dissertação, tem como contribuições: i) obtém melhores soluções que a busca local com múltiplos reinícios aleatórios, demandando um custo computacional equivalente; ii) consome menos tempo computacional em relação aos algoritmos genéticos mono-objetivo e multiobjetivo; iii) tem simples implementação.

Alguns estudos têm realizado comparações entre diferentes propostas de melhorias nos algoritmos genéticos como, por exemplo, o uso de operadores genéticos adaptados para problemas de agrupamentos (FALKENAEUR, 1996) e procedimentos de busca local para refinar os melhores indivíduos da população (BOTELHO et al., 2011). Como segundo objetivo desta pesquisa, é apresentado um estudo comparativo de 20 configurações de algoritmos genéticos, incluindo operadores genéticos adaptados para o problema CMS, identificando a melhor versão do algoritmo dentre as configurações estudadas.

Como último objetivo desta pesquisa, é realizado um estudo comparativo entre o algoritmo proposto baseado na metaheurística *Iterated Local Search*, uma busca local subida de encosta com reinícios aleatórios e duas das melhores configurações obtidas do algoritmo genético. Entre as questões avaliadas na análise comparativa das heurísticas estão: i) a eficácia (qualidade das soluções) encontrada por cada heurística, considerando-se o mesmo número de avaliações; e ii) a eficiência (tempo de execução) das heurísticas.

1.5 Organização da Dissertação

Esta dissertação está organizada em cinco capítulos. O Capítulo 1 é dedicado à introdução com a definição do problema de Clusterização de Módulos de Software (CMS) e os objetivos da pesquisa. No Capítulo 2 é apresentada uma revisão bibliográfica dos trabalhos relacionados e as métricas de avaliação mais utilizadas. As propostas de soluções para o problema CMS utilizadas no estudo comparativo, com os

parâmetros utilizados em cada heurística, são descritas com detalhes no Capítulo 3. Em seguida, o Capítulo 4 apresenta os resultados obtidos nos experimentos comparativos e avaliação dos resultados. Por último, o Capítulo 5 apresenta as considerações finais, com as conclusões, contribuições da pesquisa e propostas de trabalhos futuros.

Capítulo 2 - Revisão Bibliográfica

Este capítulo apresenta as principais abordagens e resultados dos trabalhos que aplicaram heurísticas ao problema CMS. Os trabalhos foram divididos de acordo com o uso de diferentes abordagens heurísticas.

A primeira seção descreve os trabalhos envolvendo algoritmos de busca local e algoritmos genéticos mono-objetivo. Estes trabalhos estão diretamente relacionados à proposta da dissertação, que compreende uma heurística eficiente para o problema CMS não hierárquico com formulação mono-objetivo.

Na segunda seção são apresentados os trabalhos envolvendo algoritmos genéticos multiobjetivo. Na terceira seção, são apresentados trabalhos que utilizam hiper-heurísticas. Entre as contribuições destas seções, estão os resultados obtidos com os algoritmos genéticos multiobjetivo. O algoritmo de busca local foi usado como “benchmark”, obtendo também resultados competitivos, principalmente em relação ao tempo computacional.

A quarta seção apresenta os trabalhos que abordaram o problema CMS usando a decomposição hierárquica dos módulos e métricas de similaridade. As Seções 5 e 6 apresentam estudos recentes que utilizam, respectivamente, algoritmos genéticos interativos e uma abordagem incremental para o problema CMS.

2.1 Busca Local e Algoritmos Genéticos Mono-objetivo

Entre os experimentos realizados com o problema CMS, destacam-se os efetuados pelo grupo da Drexel University (DOVAL et al., 1999, MANCORIDIS et al., 1998). Em diversos experimentos comparativos (DOVAL et al., 1999, HARMAN et al., 2002) (MITCHELL e MANCORIDIS, 2002), os resultados obtidos por um algoritmo de busca local utilizando a técnica subida de encosta com múltiplos reinícios aleatórios (*Random Restart Hill Climbing*) mostraram-se superiores aos de meta-heurísticas mais

complexas, tais como algoritmos genéticos e *Simulated Annealing*, utilizando-se o mesmo número de avaliações de soluções como critério de parada.

Baseado nestes resultados, o grupo da Drexel University desenvolveu uma ferramenta denominada Bunch (MITCHEL, 2002, MANCORIDIS et al., 1999) com a possibilidade de seleção entre uma busca local, uma heurística baseada em *Simulated Annealing* e algoritmos genéticos. A busca local do tipo subida de encosta (*Hill Climbing*) é executada para cada solução de um conjunto de soluções iniciais geradas de forma aleatória. A quantidade de soluções aleatórias a serem geradas é um parâmetro configurável na ferramenta. A relação de vizinhança utilizada é a movimentação de um módulo de um *cluster* para outro. Também é possível configurar a quantidade mínima de soluções vizinhas a serem avaliadas. Para $n = 0\%$ o algoritmo usará o método de primeira melhoria (*first-improved*), retornando o primeiro vizinho com melhor valor encontrado, enquanto para $n = 100\%$ retornará a melhor solução vizinha (*best-improved*).

Outra técnica usada para melhoria da clusterização na ferramenta Bunch é a identificação dos módulos “onipresentes”. Módulos “onipresentes” são módulos que agem como bibliotecas ou *drivers*, provendo ou consumindo serviços de vários subsistemas na aplicação. O princípio defendido é que os módulos “onipresentes” não devem ser considerados na clusterização por obscurecer a estrutura do software. Estes módulos são identificados na ferramenta por uma funcionalidade à parte (*Omnipresent Module Calculator*) e são separados em três categorias diferentes: *clients*, *suppliers* e *clients & suppliers*. Eles são considerados “onipresentes” se contiverem três vezes mais dependências que um módulo típico. Após a identificação automática pela ferramenta, os usuários podem editar a lista de módulos, adicionando ou removendo módulos onipresentes. Os módulos onipresentes constituirão subsistemas (*clusters*) distintos na aplicação, não participando no restante do processo de clusterização.

A função de avaliação utilizada na ferramenta Bunch é a *Modularization Quality (MQ)*, proposta por MITCHEL e MANCORIDIS (2002, 2006, 2007). Cabe ressaltar que a função *MQ* original (DOVAL et al., 1999) foi posteriormente modificada pelos mesmos autores. MITCHEL (2002) em sua tese de doutorado trata a função *MQ* original pelo nome *Basic MQ* e a reformulação como *Turbo MQ*. Em trabalhos posteriores os autores mencionam a função *MQ* reformulada simplesmente como *MQ*. Segundo MITCHELL a reformulação permitiu o suporte a grafos MDG ponderados, além de ser bem mais rápida que a *Basic MQ*. A função *Turbo MQ* foi utilizada na

presente dissertação e como em trabalhos recentes dentro do tema CMS será mencionada simplesmente como MQ . MITCHEL (2002) também propôs otimizar o cálculo da função MQ no contexto de busca local em que uma solução é modificada através da movimentação de um módulo de um cluster para outro. Após a movimentação, ao invés de calcular o somatório do valor do CF de cada cluster, atualiza-se o valor da função MQ pela diferença decorrida apenas do CF dos clusters modificados pela movimentação do módulo. Esta proposta foi denominada por ele de *Incremental Turbo MQ (ITurboMQ)*, e foi utilizada no presente trabalho na busca local implementada.

A função MQ realiza a avaliação considerando um balanço entre a coesão e o acoplamento dos *clusters*, recompensando *clusters* com módulos que tenham muitas dependências com outros módulos do mesmo *cluster* e penalizando aqueles com muitas dependências com módulos de outros *clusters*. O cálculo do valor de MQ é realizado através do somatório do *Cluster Factor (CF)* de cada *cluster* do software. De acordo com KÖHLER et al. (2013), CF pode ser visto como uma generalização da função conhecida como densidade relativa, que mede a qualidade de um cluster em um grafo. SIMA e SCHAEFFER (2005) provaram que o problema de decisão associado com a tarefa de encontrar os clusters ótimos, levando em consideração a medida de densidade relativa, é NP-Completo. Abaixo, é apresentada a fórmula da função MQ , para $i=1, \dots, k$, onde k é o número de *clusters*, a_i é o número de dependências entre módulos dentro do *cluster* C_i e b_i é o número de dependências que cruzam os limites do *cluster* C_i .

$$MQ = \sum_{i=1}^k CF(C_i)$$

$$CF(C_i) = 0 \quad \text{se } a_i = 0$$

$$CF(C_i) = a_i / (a_i + b_i / 2) \quad \text{se } a_i > 0$$

MAHDAVI et al. (2003) apresentam uma melhoria no algoritmo de busca local da ferramenta Bunch (MANCORIDIS et al., 1999). Em uma fase inicial, um conjunto de execuções do algoritmo de busca local é efetuado. Nesta fase uma solução parcial é construída, fixando-se os módulos que compartilham o mesmo *cluster* em $\alpha\%$ das soluções, onde α é um parâmetro definido no intervalo entre 10 e 100. Em uma segunda fase, um novo conjunto de execuções do algoritmo é realizado sobre um espaço menor de busca, uma vez que alguns módulos tiveram os *clusters* definidos na primeira fase.

Outro trabalho que utiliza uma busca local é o estudo comparativo realizado por HARMAN et al. (2005), onde os autores comparam a função MQ com uma segunda função de avaliação mono-objetivo chamada EVM (TUCKER et al., 2001), cuja definição será mostrada mais adiante. Os autores avaliam as duas funções em relação à sua capacidade de gerar distribuições de módulos em *clusters* consistentes em cenários com ruídos. A intenção é avaliar qual das funções é capaz de lidar com dependências que foram adicionadas indevidamente ou não foram removidas durante a evolução do software. O termo “ruído” é utilizado para caracterizar este possível legado de dependências, sendo gerado aleatoriamente nas instâncias utilizadas como teste antes que estas sejam submetidas ao processo de clusterização.

Os autores concluíram que a função EVM apresenta maior robustez em relação à introdução de ruídos no grafo que representa o software, embora a função MQ também seja relativamente robusta. A função EVM também tem como característica não fazer distinção entre grafos MDG ponderados e não ponderados. A diferença entre as duas funções (MQ e EVM) também diminui conforme os grafos MDG extraídos de aplicações reais aumentam de tamanho. Os estudos foram realizados com três classes de MDG: extraídos de aplicações reais, construídos aleatoriamente e MDG “perfeitos”, que são construídos automaticamente. Os MDG perfeitos⁴ consistem de grafos onde a melhor clusterização é conhecida, sendo construídos de modo que os *clusters* tenham sequencialmente tamanhos cada vez maiores, com módulos totalmente conectados dentro dos *clusters* e sem dependências com módulos de outros *clusters*. Foram realizadas comparações com grafos MDG perfeitos com tamanhos até 100 módulos. Os ruídos nos grafos foram simulados através da mutação de algumas das arestas. Uma mutação de $x\%$ em um MDG corresponde que para $x\%$ dos pares de nós, uma aresta foi adicionada ou removida.

Formalmente, a função EVM é definida como mostrado a seguir. Seja C uma clusterização com k *clusters*, tal que $C = \{C_1, \dots, C_k\}$. Sejam M_i a quantidade de módulos do i -ésimo *cluster $C_i \in C$, C_{ij} uma referência ao j -ésimo módulo do *cluster C_i e $h(C_i)$ o valor parcial para o cômputo da EVM obtida para o *cluster* C_i . Então:**

$$EVM(C) = \sum_{i=1}^k h(C_i)$$

⁴ Os grafos MDG perfeitos aqui mencionados não tem correlação com grafos perfeitos (CLAUDE, 1963) tratados em teoria dos grafos.

$$h(C_i) = \begin{cases} \sum_{a=1}^{M_i-1} \sum_{b=a+1}^{M_i} L(C_{ia}, C_{ib}) & \text{se } M_i > 1 \\ 0 & \text{se } M_i \leq 1 \end{cases}$$

$$L(C_{xy}, C_{pq}) = \begin{cases} 1 & \text{se existir dependência entre os módulos Cxy e Cpq} \\ -1 & \text{caso não exista dependência entre os módulos Cxy e Cpq} \end{cases}$$

Outro experimento comparativo envolvendo algoritmos genéticos mono-objetivo para o problema CMS foi apresentado por PRADITWONG (2011), que utiliza um operador genético de recombinação adaptado para o problema CMS. A adaptação proposta é a junção das ideias de operadores genéticos utilizados em problemas de agrupamento, como o BPRX (*Bin Packing Crossover with Replacement*) proposto por FALKENAUER (1996), com uma nova representação do cromossomo para o CMS proposta por HARMAN et al. (2002). Os resultados do estudo (PRADITWONG, 2011) indicam que o algoritmo genético que utiliza a representação com os agrupamentos (*Grouping Genetic Algorithm*) retorna soluções melhores do que as soluções obtidas com a representação predominantemente utilizada (*Grouping Number Encoding*), na qual cada índice de um vetor ou de uma sequência de caracteres representa o módulo e o conteúdo representa o *cluster* que contém o módulo.

DIAS e OCHI (2003) apresentam sete versões adaptadas do algoritmo genético tradicional (DOVAL et al., 1999) para aumentar a performance e melhorar a qualidade das soluções obtidas para o problema de particionamento de grafos não ponderados. Estas adaptações foram avaliadas através de estudos experimentais que utilizaram instâncias geradas artificialmente. Os autores não relacionaram os estudos com o problema CMS, embora a representação subjacente seja a mesma. A função de avaliação utilizada foi a função *Basic MQ*, originalmente proposta por DOVAL et al. (1999). Entre as adaptações sugeridas, destacam-se os métodos a seguir:

- a) Inserção de um procedimento de busca local após a aplicação do operador de mutação. O procedimento de busca é aplicado apenas no indivíduo mais apto de cada geração, devido ao custo computacional no procedimento de avaliação, onde o novo indivíduo precisa ser reavaliado. Para cada um dos módulos, o procedimento de busca identifica o *cluster* no qual o módulo compartilha o maior número de dependências. Se este *cluster* for diferente do *cluster* atual do módulo, é realizada avaliação da solução considerando a movimentação do módulo para o *cluster* identificado. Caso haja aumento do *fitness* a troca é efetivamente realizada;

- b) Um método para calibrar o operador de mutação, permitindo ajustar progressivamente o número máximo de *clusters* modificados pelo operador de mutação a medida que as soluções vão convergindo;
- c) Um procedimento para diversificação das soluções, onde a melhor solução da geração corrente é replicada n vezes. Em cada uma destas réplicas, considerando uma janela de tamanho aleatório, os genes têm seus valores trocados aleatoriamente, com o objetivo de realizar uma busca nesta região do espaço de soluções.

Posteriormente, DIAS (2004) estendeu estes experimentos comparando diferentes adaptações do algoritmo genético na versão clássica (DOVAL et al., 1999) com 135 adaptações de algoritmos genéticos resultantes da combinação de diferentes técnicas. Os algoritmos genéticos na versão clássica foram denominados pelo autor por AGT (“Algoritmo Genético Tradicional”), enquanto os algoritmos genéticos com as técnicas combinadas foram chamados de AEH (“Algoritmo Evolutivo Híbrido”). Foram utilizadas vinte instâncias artificiais, com grafos variando entre 10 e 100 vértices, e dois tipos de topologia: anel e estrela. As versões do AEH consistem do resultado da combinação das seguintes técnicas:

1. Taxa de mutação igual a 0,0040;
2. Calibração do número de *clusters* no operador de mutação;
3. Seleção por torneio de 2 indivíduos;
4. Seleção por torneio de 4 indivíduos;
5. Operador de recombinação por agrupamentos (GGA ou *Grouping Genetic Algorithm*) (FALKENAUER, 1996);
6. Diversificação após 10 gerações sem evolução média da população;
7. Diversificação após 25 gerações sem evolução média da população;
8. Aplicação do procedimento de busca local no melhor indivíduo de cada geração.

Os resultados indicam que os algoritmos genéticos com as técnicas combinadas (AEH) são superiores ao algoritmo genético na versão clássica. As melhores versões sempre possuem as seguintes propostas combinadas: alteração na taxa do operador de mutação, seleção por torneio, inclusão do procedimento de diversificação e inclusão do procedimento de busca local. O AEH que retorna a melhor média (do número de iterações em que encontra a melhor solução) é o AEH13578 (combinação das técnicas 1, 3, 5, 7 e 8).

Outro experimento comparativo visando aumentar o desempenho do algoritmo genético tradicional é apresentado por BOTELHO et al. (2011). Os autores comparam diferentes algoritmos genéticos utilizando a técnica de reconexão de caminhos (GLOVER e KOCHENBERGER, 2003), um procedimento de busca local e um método construtivo na geração da população inicial, que considera o peso das arestas em vez de gerar soluções iniciais de forma totalmente aleatória. Os resultados indicam que as adaptações utilizando o procedimento adicional de busca local são superiores às versões utilizando reconexão de caminhos e o procedimento construtivo.

Um estudo que tem uma abordagem distinta dos algoritmos evolutivos citados até então é o estudo de SEMAAN et al. (2011), que propõe a utilização da meta-heurística *Iterated Local Search* (ILS) para a resolução do problema de agrupamento em sistemas orientados a objetos. Distingue-se do problema CMS aqui apresentado ao tratar grafos MDGs ponderados e avaliar os resultados através da função *MQ* original (DOVAL et. al, 1999) com adaptação para o cálculo de MDGs ponderados. Na proposta de Semaan, a meta-heurística utiliza três procedimentos de perturbação, selecionados de forma aleatória:

- a) Migração aleatória: cada módulo possui uma probabilidade de migrar para outro *cluster*;
- b) Explosão: um cluster é selecionado aleatoriamente e cada módulo do cluster selecionado formará um novo cluster;
- c) Divisão: um cluster é selecionado aleatoriamente e seus módulos serão distribuídos em dois novos clusters.

SEMAAN et al. (2011) utilizaram 10 instâncias artificiais com 10 a 100 vértices neste estudo. O algoritmo proposto baseado em ILS obteve melhores resultados, alcançando o alvo médio (média obtida através de resultados na literatura (BOTELHO et al., 2011, SEMAAN e OCHI, 2007) em menos tempo que o melhor algoritmo genético de BOTELHO et al. (2011), que utiliza o operador de recombinação *single-point crossover* e o procedimento de busca local.

2.2 Algoritmos Genéticos Multiobjetivo

Os algoritmos genéticos multiobjetivo utilizam formulações que consideram múltiplos objetivos possivelmente conflitantes. Representando separadamente os diferentes objetivos, estes algoritmos evitam a resolução automática de conflitos que podem vir a acontecer na formulação mono-objetivo ao procurar balancear objetivos distintos (como baixo acoplamento e alta coesão) em uma única equação. A otimização multiobjetivo comumente utilizada na área de SBSE se baseia em fronteiras de Pareto, que são formadas com base na relação de dominância definida a seguir.

$$F(\bar{x}_1) > F(\bar{x}_2) \Leftrightarrow \forall i. f_i(\bar{x}_1) \geq f_i(\bar{x}_2) \wedge \exists i. f_i(\bar{x}_1) > f_i(\bar{x}_2)$$

A solução x_1 é dita dominante sobre outra solução x_2 se pelo menos um dos objetivos individuais retorna um valor melhor em x_1 quando comparado a x_2 e nenhuma outra função retorna valores piores em x_1 do que em x_2 . Se em um conjunto X de soluções, nenhum elemento é dominante sobre uma solução y fora do conjunto X , então a solução y é não dominada por X . Uma fronteira de Pareto é formada pelo conjunto de soluções não dominadas por qualquer das soluções percorridas no espaço de busca até um dado momento, servindo para guiar a busca por melhores soluções.

Entre os estudos que abordam o CMS com otimização multiobjetivo é possível citar os trabalhos de PRADITWONG et al. (2011) e BARROS (2012), que utilizam as formulações MCA e ECA para o problema. A formulação MCA (*Maximizing Cluster Approach*) utiliza o seguinte conjunto de objetivos: a) maximizar a soma dos pesos das arestas internas entre módulos, para todos os clusters; b) minimizar a soma dos pesos das arestas entre módulos de clusters distintos; c) maximizar o número de clusters; d) maximizar o valor da função *MQ* (*Modularization Quality*); e e) minimizar o número de clusters que contém um único módulo. A formulação ECA (*Equal-size Cluster Approach*) é composta por cinco objetivos, sendo os primeiros quatro objetivos idênticos aos da formulação MCA. No quinto objetivo, ao invés de minimizar o número de clusters com somente um módulo, ECA procura minimizar a diferença entre o número máximo e mínimo de módulos nos clusters de um agrupamento.

PRADITWONG et al. (2011) apresentam um estudo comparativo entre três heurísticas: uma busca local do tipo subida de encosta com reinícios aleatórios (*Random Restart Hill Climbing*) usando a função mono-objetivo *MQ* e dois algoritmos genéticos multiobjetivo que utilizam respectivamente as formulações ECA e MCA. Neste estudo

foram utilizadas 17 instâncias extraídas de aplicações reais com até 200 módulos, incluindo instâncias representadas como MDG ponderados e não ponderados. Os algoritmos genéticos foram baseados no algoritmo *Two-Archive*, proposto por PRADITWONG e YAO (2006). Eles foram configurados com operador de recombinação *single-point crossover* (taxa de cruzamento de 80% para instâncias onde $n < 100$ e 100% as demais), operador de mutação *single-point mutation* (com taxa de mutação fixa igual a $0,004 \cdot \log_2 N$), operadores de seleção por torneio (*binary tournament*) e por roleta (*roulette wheel*), população igual a $10 \cdot n$ e número máximo de gerações igual a $200 \cdot n$, onde n é o número de módulos da instância sendo analisada. Os resultados dos estudos reportados por PRADITWONG et al. (2011) indicam que, em grafos ponderados, as formulações multiobjetivo retornam resultados de melhor qualidade que a busca local baseada na função MQ , embora os algoritmos genéticos consumam duas ordens de magnitude mais de tempo computacional para produzir seus resultados. Estes resultados foram posteriormente confirmados por KISHORE et al. (2012), que replicaram o estudo de PRADITWONG et al. (2011) utilizando as mesmas instâncias. Além disso, a busca local retornou melhores resultados em três das sete instâncias utilizando MDG não ponderados, enquanto nas 10 instâncias representadas por MDG ponderados o algoritmo genético utilizando a formulação multiobjetivo ECA obteve melhores valores de MQ .

BARROS (2012) realizou experimentos comparando três configurações distintas de um algoritmo genético multiobjetivo (NSGA-II – *Non-Dominating Sorting Genetic Algorithm* (DEB e KALYANMOY, 2001) e uma busca local do tipo subida de encosta com reinício aleatório. As configurações dos algoritmos genéticos multiobjetivo utilizadas no trabalho são: (a) NSGA-II utilizando a formulação multiobjetivo ECA; (b) NSGA-II utilizando a formulação multiobjetivo ECA sem incluir o objetivo de maximizar o valor da função MQ (usado como objetivo da ECA original); e (c) NSGA-II utilizando a formulação multiobjetivo ECA sem incluir o objetivo de maximizar a função MQ e adaptada para maximizar a EVM . Os resultados obtidos no estudo indicam que suprimir a função MQ ajuda a encontrar soluções em menos tempo, particularmente para as instâncias maiores, e sem perda de qualidade. Para as instâncias consideradas, a troca da função MQ pela função EVM aumentou a eficiência e a eficácia em relação à formulação multiobjetivo ECA original.

Outra referência que apresenta algoritmos genéticos aplicados ao problema CMS é o survey de RÄIHÄ (2010). RÄIHÄ fez o levantamento dentro da linha de pesquisa

SBSE e dentro do tema *software design*, deste modo também abrangendo trabalhos vinculados ao tema CMS. Rähä apresentou informações tais como: entradas utilizadas nos métodos de buscas, diferentes codificações de cromossomos, operadores de mutação e operadores de recombinação dos algoritmos genéticos. As representações de cromossomo mais comuns mencionadas neste trabalho foram vetores de inteiros, string de inteiros e MDG.

2.3 Hiper-heurísticas

KUMARI e SRINIVAS (2012) propõem o uso de hiper-heurísticas (COWLING et al., 2001) para resolver o problema CMS. Os autores definem o conceito de hiper-heurística conforme a seguir: "as hiper-heurísticas gerenciam a escolha de qual heurística de mais baixo nível deveria ser aplicada em um determinado momento, dependendo das características da heurística e da região do espaço de busca sob exploração".

Os autores propõem o uso de algoritmos evolutivos (MHypEA) multiobjetivo combinando doze heurísticas de baixo nível, que incluem diferentes métodos de seleção, recombinação e mutação. As heurísticas são resultado da combinação de diferentes operadores genéticos, tais como:

- a) Operadores de seleção: *rand* e *rand-to-best*. No primeiro, os dois pais são selecionados aleatoriamente, enquanto no segundo um dos pais é selecionado aleatoriamente e o outro é selecionado do conjunto elite;
- b) Operadores de recombinação: *uniform-crossover*, *hybrid-crossover* (combina *uniform-crossover* com *single-point-crossover*) e *hybrid-crossover-2* (combina *uniform-crossover* com *two-points-crossover*);
- c) Operadores de mutação: *copy* (dois genes são selecionados aleatoriamente e o segundo é copiado para o primeiro) e *exchange* (dois genes são selecionados aleatoriamente e trocados de posição).

As heurísticas são selecionadas de acordo com pesos, atribuídos ao longo das gerações. Se uma determinada heurística gera uma população com maior desempenho, seu peso é aumentado, aumentando suas chances de seleção em novas rodadas. Caso contrário, a heurística pode ter seu peso diminuído, reduzindo sua probabilidade de seleção em uma próxima iteração.

Foram utilizadas seis instâncias de teste, variando entre 20 e 174 módulos. A heurística MHypEA se mostrou promissora ao equilibrar a exploração e pesquisa do espaço de busca, obtendo melhores valores para a função MQ do que o algoritmo genético multiobjetivo *Two-Archive* de PRADITWONG e YAO (2006) em todas as seis instâncias, embora a diferença do valor de MQ obtidos pelas duas técnicas não seja alta. Considerando-se as melhores soluções obtidas em cada heurística, a diferença percentual foi sempre menor que 1% (a maior diferença foi na instância “Rcs”, onde o algoritmo *Two-Archive* obteve valor de MQ igual a 2,239 e o MHypEA obteve 2,242). Outra vantagem observada pelos autores é que a hiper-heurística consome aproximadamente um vigésimo do tempo computacional gasto pelo algoritmo multiobjetivo.

2.4 Clusterização hierárquica

Embora grande parte das publicações trate o problema CMS sob o ponto de vista não hierárquico, alguns autores tratam especificamente da visualização do problema CMS em forma hierárquica, levando-se em consideração o melhor modo de decompor o software em uma árvore de componentes aninhados.

LUTZ (2001) denomina HMD (*Hierarchical Modular Decomposition*) a estrutura hierárquica de módulos. Ao contrário de outras abordagens, que levam em conta o acoplamento e a coesão na avaliação da distribuição de módulos em *clusters*, este trabalho têm como objetivo obter a decomposição mais simples, isto é, a estrutura hierárquica mais curta (*Minimum Description Length Principle*) (RISSANEN, 1978) que compreenda todos os módulos. Segundo o autor, a aplicação do princípio *Minimum Description Length Principle* e a escolha da estrutura mais curta faz com que se obtenha, na prática, as características desejadas de baixo acoplamento e alta coesão. Lutz propõe uma métrica baseada no mesmo princípio (variante de uma métrica proposta anteriormente por WOOD (1998) e adaptada para possibilitar comparações de soluções hierárquicas). No algoritmo genético *distributed “steady-state”* (COLLINS e JEFFERSON, 1991), a população é distribuída espacialmente em um grid toroidal e os indivíduos só podem ser recombinados com outros em sua vizinhança. O grid em forma toroidal pode ser obtido a partir de um grid bidimensional de dimensões $N \times N$ unindo os indivíduos da extremidade, fazendo com que cada indivíduo tenha a mesma quantidade de vizinhos (ALBA e DORRONSORO, 2008). O operador de recombinação é baseado

no operador *tree-crossover* (KOZA, 1992), com um passo adicional para a concatenação dos dois indivíduos representado a estrutura HMD. A mutação pode realizar três tipos de operação: i) mover aleatoriamente um nó (pode ser um módulo ou *cluster* contendo módulos) para outra posição aleatória na árvore, com a condição de não formar ciclos; ii) a partir de um *cluster c'* selecionado aleatoriamente criar um novo *cluster c''* (que estará contido em *c'*), formado de um subconjunto de módulos pertencentes ao *cluster c'*; e iii) selecionar aleatoriamente um *cluster c'* que não seja do primeiro nível, fazendo com que o *cluster* que o contém, passe a conter todos os módulos e *clusters* antes contidos em *c'*. O algoritmo genético foi executado sobre três exemplos. Entre os resultados apresentados, o autor conclui que o HMD obtido tem seus componentes melhor organizados, além de logicamente melhor que a organização encontrada por WOOD (1998) para a instância *traffic lights*.

ANQUETIL et al. (1999) avalia a performance de quatro algoritmos de clusterização aglomerativos hierárquicos. Os algoritmos aglomerativos hierárquicos partem dos itens individuais, reunindo-os em pequenos clusters segundo um determinado critério. Os pequenos clusters vão sendo reunidos em clusters maiores, sucessivamente, até que se obtenha um cluster contendo todos os itens. Os algoritmos avaliados se diferenciam pelo modo como calculam a distância de um novo cluster em relação aos outros. Foram avaliados quatro algoritmos distintos: *single linkage*, *complete linkage*, *weighted average linkage* e *unweighted average linkage*. Os autores utilizam as funções *Precision* e *Recall* para comparar diferentes particionamentos do grafo de módulos que representa um software. *Precision* informa a porcentagem de pares de *clusters* encontrados no particionamento obtido por um algoritmo que também se encontram no mesmo cluster de um particionamento fornecido previamente. De modo similar, *Recall* informa a porcentagem de pares de *clusters* de um particionamento fornecido que também se encontram no particionamento encontrado por um algoritmo. Entre outras conclusões, os autores relatam que não encontraram diferença significativa para estas métricas entre os algoritmos hierárquicos e não hierárquicos. O particionamento padrão fornecido é denominado no estudo de “expert partition”. As “expert partitions” utilizadas no estudo, foram coletadas a partir de três sistemas (Linux, Mosaic e um sistema legado), com os módulos e clusters definidos de acordo com a estrutura de diretórios no qual os sistemas estavam organizados.

MITCHELL e MANCORIDIS (2001) realizaram um estudo comparando quatro diferentes medidas de similaridade. Entre as medidas comparadas, duas foram utilizadas

em estudos prévios, a MoJo (TZERPOS e HOLT, 1999) e Precision, enquanto duas novas medidas foram propostas pelos autores: EdgeSim e MeCl. A medida MoJo é baseada na quantidade de movimentos e uniões de clusters necessárias para transformar um agrupamento de clusters obtido em outro agrupamento. A medida Precision/Recall (ANQUETIL et al., 1999) baseia-se nos pares de agrupamentos localizados no mesmo *cluster*. EdgeSim é uma medida proposta pelos autores que utiliza tanto a posição dos módulos nos *clusters* quanto as dependências entre módulos para o cálculo de similaridade. *Merge Clusters* (MeCl) calcula a distância entre duas soluções através de divisões e uniões dos *clusters* necessárias para reproduzir uma solução. Entre os resultados apresentados por MITCHELL e MANCORIDIS (2001) estão: i) as quatro medidas se comportaram de forma consistente em dez instâncias examinadas; ii) a remoção dos módulos onipresentes melhorou as medidas de similaridade (em média 12,4% de melhoria para Precision/Recall, 9,7%, para MoJo, 13,6% para EdgeSim e 7,3% para MeCl); e iii) as medidas EdgeSim e MeCl, que ao contrário das outras abordagens consideram as dependências entre os módulos, obtiveram resultados menos variáveis que as medidas MoJo e Precision/Recall.

Em relação à função MoJo, WEN e TZERPOS (2004) propuseram a MoJoFM (MoJo eFfectiveness Measure) visando corrigir algumas deficiências da MoJo. As deficiências mencionadas são: i) o algoritmo original para calcular a MoJo poderia ser aperfeiçoado (WEN e TZERPOS, 2003); ii) é uma medida não simétrica, de modo que a quantidade de movimentos e uniões para transformar a solução A em B é diferente do número para transformar B em A; e iii) propor uma medida normalizada e mais acurada que a MoJo. MoJoFM tem a seguinte fórmula:

$$MoJoFM(A, B) = 1 - \frac{mno(A, B)}{\max(mno(\forall A, B))}$$

onde A e B são duas partições. $mno(A, B)$ é o número mínimo de movimentos ou uniões para transformar a partição A na partição B (WEN e TZERPOS, 2003) e $\max(mno(\forall A, B))$ é a máxima distância possível para transformar uma partição A na partição B . Os resultados indicaram uma melhoria no desempenho em relação a MoJo e que a nova métrica pode ser usada para avaliar algoritmos de clusterização de software.

Outro algoritmo de clusterização hierárquico é o ACDC (*Algorithm for Comprehension-Driven Clustering*) (TZERPOS et al., 2000). Ao contrário de outros algoritmos que estão baseados em informações estruturais e tem como prioridade

satisfazer métricas tais como baixo acoplamento e alta coesão, o algoritmo ACDC tem como objetivo uma saída facilmente compreensível, garantindo que os clusters sigam padrões familiares, além de nomear os clusters de forma inteligente. Outro fator considerado no algoritmo é manter a quantidade de módulos no cluster em um número gerenciável (por exemplo, em torno de vinte módulos). Foram realizados experimentos com duas instâncias reais. Entre as conclusões estão que o algoritmo ACDC pode ser um algoritmo de clusterização eficaz e um bom candidato para projetos de engenharia reversa. Resultados comparativos posteriores (SHTERN e TZERPOS, 2004) entre o algoritmo ACDC e a ferramenta Bunch indicam que o ACDC encontra particionamentos hierárquicos superiores aos encontrados pela ferramenta Bunch, embora esta seja mais eficaz em particionamentos não hierárquicos.

Recentemente, HALL e MCMINN (2012) realizaram uma análise da ferramenta Bunch na abordagem hierárquica. A ferramenta Bunch foi utilizada para produzir soluções hierárquicas através de sucessivas clusterizações. Foi realizada uma abordagem *bottom-up*, onde cada clusterização agrupou os *clusters* produzidos anteriormente, ou seja, os *clusters* funcionando como módulos na nova clusterização. A realização da clusterização hierárquica através de seguidas clusterizações faz com que os diferentes ramos das árvores de componentes tenham profundidade semelhante, produzindo uma árvore balanceada. Outra questão é que o *fitness* das clusterizações nos últimos níveis é alterado pelo *fitness* obtido nas clusterizações dos níveis anteriores. Os resultados do estudo indicam uma melhoria de até 30% através da redução da estratificação em camadas dos diversos níveis hierárquicos produzidos pela clusterização. Os autores apresentaram um operador de mutação formando uma árvore de componentes não balanceada. A estratégia de remoção aleatória de módulos (cem vezes a cada resultado produzido pela ferramenta) acarretou melhoria no valor de *MQ* para as cinco instâncias testadas, mostrando que os resultados com árvores não balanceadas eram superiores que os resultados com profundidade de ramos uniforme.

AMOUI et al. (2006) tratam o problema CMS representando a reorganização do software em forma de padrões de projeto (*GOF Patterns*)(GAMMA et al., 1995). A avaliação da solução é realizada por uma versão normalizada da métrica *Distance from the Main Sequence (D)* (MARTIN, 2000). A métrica utiliza os conceitos de acoplamento eferente e aferente. Acoplamento eferente é a quantidade de classes de quem uma classe depende. Acoplamento aferente é a quantidade de classes dependentes de uma classe. A fórmula da métrica é a seguinte:

$$D = \frac{|A+I-1|}{\sqrt{2}}$$

Onde:

$$I = \frac{\text{Acooplament o Eferente}}{\text{Acooplament o Eferente} + \text{Acooplament o Aferente}}$$

$$A = \frac{\text{Classes Abstratas}}{\text{Total Classes}}$$

São utilizados algoritmos genéticos para encontrar as soluções que representam a aplicação de uma seqüência de padrões de projeto. Cada cromossomo codifica uma seqüência de transformações e parâmetros específicos de acordo com cada padrão de projeto. Seja um supergene um gene de tamanho variável. Cada supergene do cromossomo é composto de três tipos de informações: um número representando o padrão de projeto, o número do pacote (cluster) e as classes em que o padrão é aplicado. Os operadores de recombinação (*crossover*) utilizados são dois: a) o primeiro operador utiliza um *single-point-crossover* aplicado a um ponto selecionado aleatoriamente ao nível do supergenes, trocando os supergenes acima deste ponto, isto é, mantém a estrutura interna dos supergenes inalterada; b) o segundo também aplica o *single-point-crossover*, porém este é aplicado em dois supergenes de dois cromossomos distintos, gerando um novo padrão de projeto (supergene) a partir da combinação dos dois supergenes selecionados. O operador de mutação utilizado seleciona aleatoriamente um supergene e modifica um número aleatório de genes dentro do supergene. Foi desenvolvido um framework (*Automatic Design Enhancer*), com duas versões com os diferentes operadores genéticos, aplicados a uma instância real (aplicação denominada *Gold Parser*). Entretanto não foram apresentados resultados comparativos em relação a outras heurísticas.

2.5 Algoritmos Genéticos Interativos (AGI)

BAVOTA et al. (2012) propõem o uso de algoritmos genéticos interativos (AGI) para abordar o CMS, integrando o conhecimento dos desenvolvedores no processo de clusterização. Nos AGI, a função objetivo é ajustada de acordo com a avaliação humana, conforme as gerações evoluem no algoritmo genético. Os autores comparam o AGI com um algoritmo genético utilizando a função multiobjetivo MCA

(PRADITWONG et al., 2011). Uma das variantes do AGI está na Figura 1. A função de aptidão do AGI é composta pela métrica MQ e uma penalidade de ajuste, fornecida de acordo com o *feedback* dos desenvolvedores. Os resultados indicam que os AGI podem fornecer distribuições de módulos em *clusters* mais significativas para os desenvolvedores e retornam clusterizações de melhor qualidade que os AG não interativos.

```
Algoritmo R-IGA: IGA com feedback de pares de componentes  
1 Para i de 1 até nIterações faça  
2     Evolua o AG por nGerações  
3     Selecione a solução com maior valor de  $MQ$   
4     Para j de 1 até nFeedbacks faça  
5         Selecione dois componentes  $c_i$  e  $c_j$   
6         Perguntar ao desenvolvedor se eles devem estar juntos ou separados  
7     Fim-para  
8     Modificar a solução de acordo com os feedbacks  
9     Gerar uma nova população usando como ponto de partida a solução corrigida  
10 Fim-Para  
11 Fim-Algoritmo
```

Figura 1. Algoritmo R-IGA (Bavota et al., 2012)

HALL et al. (2012) apresentam o algoritmo SUMO (*Supervised Modularization*) visando complementar outras abordagens não interativas. Este algoritmo apresenta particionamentos hipotéticos ao usuário, que concordará com algumas das relações e discordará de outras. As relações correspondem a pares de módulos, divididas entre os conjuntos Rel+ (pares que devem permanecer no mesmo *cluster*) e Rel- (pares que devem pertencer a *clusters* distintos). As correções do usuário são integradas ao processo de clusterização, reduzindo o espaço de busca pela distribuição ideal de módulos e conduzindo a novos particionamentos mais refinados. O processo é repetido até que um resultado satisfatório seja atingido.

SIMONS et al. (2012) utilizam algoritmos evolutivos interativos para aprimorar o modelo do software. Cada indivíduo da população corresponde a um modelo do software. Em cada indivíduo as classes representam os *clusters* e os métodos e atributos representam os módulos a serem distribuídos pelas classes. Neste trabalho eles propõem novas métricas para quantificar a elegância e simetria do software, medindo o quanto estas métricas estão relacionadas com as melhores avaliações dos desenvolvedores. Foram usadas as seguintes métricas:

- a) *Numbers among classes* (NAC) elegance: desvio padrão do número de atributos e métodos entre as classes;
- b) *External couples elegance* (EC): desvio padrão do número de dependências entre métodos de classes distintas;
- c) *Internal uses elegance* (IU): desvio padrão dos usos “internos” nas classes. A quantidade de usos “internos” em uma classe corresponde à quantidade de atributos da classe que um método utiliza, considerando atributos internos da classe;
- d) *Attributes to methods ratio* (ATMR): é o desvio padrão da razão entre o número de atributos e métodos internamente às classes.

Os resultados demonstram que as métricas NAC, EC e ATMR estão correlacionadas com as melhores avaliações dos desenvolvedores, sugerindo que a uniformidade de distribuição dos componentes é um fator percebido e recompensado. A métrica NAC foi utilizada em outro trabalho posterior (BARROS e FARZAT, 2013) aplicado ao problema CMS.

2.6 Incremental Search-Based-Software Engineering (ISBSE)

BARROS e FARZAT (2013) realizaram um estudo sobre a evolução do Apache Ant, uma ferramenta para *build* de aplicações desenvolvida em Java. Os autores observaram que a simplicidade do modelo original foi perdida ao longo das sucessivas manutenções e adição de novas funcionalidades. A partir desta observação os autores estudaram a aplicação de técnicas heurísticas (SBSE), para verificar a possibilidade de recuperar partes do modelo original. Foi utilizado no estudo a busca local do tipo subida de encosta com reinícios aleatórios. As métricas utilizadas foram *EVM*, *MQ*, NAC, LCOM (HENDERSON-SELLERS, 1996) e CBO (CHIDAMBER e KEMERER, 1994). Entre as conclusões estão que as técnicas e métricas produzem modelos complexos, com aumento significativo no número de pacotes (clusters). Uma proposta em face deste resultado é unificar a clusterização de módulos de software com métodos para seleção dos módulos pelos usuários, restringindo a extensão das mudanças no software.

Posteriormente, BARROS (2013) introduz o conceito *Incremental Search-Based-Software-Engineering (ISBSE)*, sugerindo pequenas mudanças pontuais no software, em vez de uma única e extensa mudança. Entre as vantagens da abordagem

incremental em SBSE, e para o problema CMS em específico, são citadas (BARROS, 2013): a) um número máximo de modificações é previamente definido, evitando extensas perdas na base de conhecimento da equipe de desenvolvimento; b) a adoção de heurísticas e processos automatizados na refatoração do software tende a aumentar, posto que os desenvolvedores não vão entender estas atividades como destrutivas em relação ao seu trabalho original; e c) possibilita a coleta de intenções e percepções dos usuários, que podem posteriormente ser utilizadas em processos de busca interativos, tais como AGI.

Foram utilizadas no experimento 32 instâncias de aplicações reais, variando entre 26 e 299 módulos. Foram aplicadas as funções de avaliação *MQ* e *EVM*, tendo a função *MQ* se saído melhor com uma média percentual de 46%. Também foi observado que houve um alto custo por restringir a busca em uma sequência de pequenas execuções, na forma de um número adicional de iterações e movimentações de módulos, que não seriam realizadas em uma única execução sem restrições.

2.7 Considerações Finais

Diversas abordagens heurísticas para encontrar boas soluções para o problema CMS têm sido propostas há vários anos. Entretanto, o problema continua atual, bem como a busca por métodos automatizados e eficientes que auxiliem os desenvolvedores nas tarefas de manutenção e evolução do software e contribuam para a garantia de qualidade. Este capítulo apresentou os trabalhos relacionados ao tema CMS sob o ponto de vista de diferentes possibilidades de pesquisa: formulação mono-objetivo x formulação multiobjetivo, abordagem hierárquica x abordagem não hierárquica e abordagem interativa x não interativa. Também foram apresentados estudos comparativos voltados para métricas de qualidade e similaridade que avaliam as soluções encontradas pelos algoritmos automáticos.

Os principais estudos baseados na formulação mono-objetivo utilizam algoritmos genéticos e busca local com reinícios aleatórios. Os estudos envolvendo algoritmos genéticos demonstram que a utilização de operadores genéticos adaptados para problemas de clusterização e o uso de um procedimento de busca local possibilitam obter melhores soluções que algoritmos genéticos utilizando operadores “tradicionais” (PRADITWONG, 2011, DIAS e OCHI, 2003). Entretanto, estes estudos não apresentaram comparações em relação à busca local. Por outro lado, existem estudos

com algoritmos genéticos (DOVAL et al., 1999, HARMAN et al., 2002) em que são apresentados resultados onde a busca local com múltiplos reinícios aleatórios se mostrou superior para instâncias extraídas de aplicações reais.

O próximo capítulo é dedicado a apresentação do algoritmo proposto, baseado na meta-heurística *Iterated Local Search*, para tratar o Problema de Clusterização de Módulos de Software. Apresenta-se também o detalhamento de outras heurísticas implementadas para serem avaliadas no experimento computacional detalhado no Capítulo 4.

Capítulo 3 - Iterated Local Search Aplicado ao Problema de Clusterização de Módulos de Software

Como mostrado no capítulo anterior, é vasta a literatura sobre heurísticas para o problema CMS. Os estudos mais recentes têm se concentrado em algoritmos genéticos multiobjetivo e algoritmos genéticos interativos. Os resultados envolvendo algoritmos genéticos multiobjetivo aplicados a grafos MDG ponderados são promissores, mas possuem um custo computacional bem superior ao da busca local com múltiplos reinícios aleatórios implementada por PRADITWONG et al. (2011).

Nos estudos envolvendo a formulação mono-objetivo, o algoritmo de busca local subida de encosta com reinícios aleatórios têm se mostrado superior a algoritmos mais complexos. Outro estudo que apresenta resultados relevantes é o de PRADITWONG (2011), que utiliza um operador de recombinação baseado em agrupamentos. No entanto, o autor não apresentou uma comparação dos seus resultados com o algoritmo de busca local subida de encosta, mas somente com outro algoritmo genético utilizando o operador de recombinação *single-point-crossover*.

O presente trabalho propõe uma heurística baseada na meta-heurística *Iterated Local Search* (ILS) (LOURENÇO et al., 2001) para o CMS. Para avaliar a eficácia (em relação à qualidade das soluções encontradas) e a eficiência (custo computacional do processo de busca) do algoritmo proposto, duas abordagens heurísticas mono-objetivo previamente aplicadas ao CMS foram implementadas, a saber: uma busca local com múltiplos reinícios e dois algoritmos genéticos usando duas diferentes representações. Neste capítulo apresentam-se a meta-heurística ILS, o algoritmo proposto e as abordagens heurísticas da literatura implementadas neste trabalho.

3.1 A Meta-heurística Iterated Local Search

A meta-heurística *Iterated Local Search* (ILS) (LOURENÇO et al. 2001) aplica uma busca local sobre uma solução inicial. Quando a busca atinge um ótimo local, é aplicado um método de perturbação sobre esta solução e a busca local é reiniciada a

partir da solução modificada pela perturbação. Deste modo, ILS busca obter soluções de melhor qualidade se comparadas com as soluções geradas por repetições aleatórias da mesma busca local.

A meta-heurística ILS possui quatro componentes principais: (i) processo de geração da solução inicial, (ii) método de busca local, (iii) método de perturbação e (iv) critério de aceitação de uma nova solução. Estes componentes serão apresentados em maiores detalhes nos próximos parágrafos. A estrutura geral da meta-heurística ILS pode ser visualizada na Figura 2.

```
Procedimento Iterated Local Search  
1   s0 = gerar solução inicial  
2   s* = buscaLocal(s0)  
3   Repita  
4       s' = perturbação(s*, histórico)  
5       s*' = buscaLocal(s')  
6       s* = critérioDeAceitação(s*, s*', histórico)  
7   Até que critério de parada satisfeito  
8   Fim
```

Figura 2. Pseudocódigo da meta-heurística ILS

O primeiro passo (linha 1) é a geração de uma solução inicial válida, que pode ser criada de forma aleatória ou construída a partir de uma heurística rápida, por exemplo, através de uma heurística construtiva gulosa. De modo geral, a utilização de uma heurística gulosa adequada tem como vantagem retornar soluções iniciais de melhor qualidade do que a geração aleatória (STÜTZLE, 1998), exigindo um menor número de passos na busca local para melhoria da solução e, conseqüentemente, menos tempo computacional.

A estratégia da heurística gulosa para geração da solução inicial também é a recomendada por LOURENÇO et al. (2001), com a ressalva que os benefícios da geração da solução inicial tendem a diminuir quanto mais demorada é a execução da meta-heurística ILS. Por outro lado, se a melhoria da solução continuar constante conforme se aumenta o tempo de execução, pode ser um indicativo de que os métodos de perturbação e aceitação utilizados não são adequados para o problema.

A Figura 3 ilustra como a meta-heurística ILS percorre o espaço de busca. Ao atingir um mínimo local s^* , ILS aplica uma perturbação em s^* obtendo uma nova solução s' . Em seguida, uma nova busca local é realizada a partir de s' , escapando da

região que conduziria ao mesmo mínimo local e encontrando um novo mínimo local $s^{*'}$. A escolha do método de perturbação (linha 4) é importante para o bom funcionamento do algoritmo baseado na meta-heurística ILS. Uma perturbação pequena fará com que a nova busca fique restrita a mesma região, recaindo sobre o mesmo ótimo local, enquanto uma perturbação muito grande fará com que ela se comporte como uma busca local com reinício aleatório. A intensidade da perturbação está relacionada à quantidade de componentes da solução que são modificados. Esta intensidade tanto pode ser fixa quanto variar durante a execução da meta-heurística. Por exemplo, pode ser interessante aumentar a intensidade caso a busca fique restrita a uma mesma região por diversas iterações.

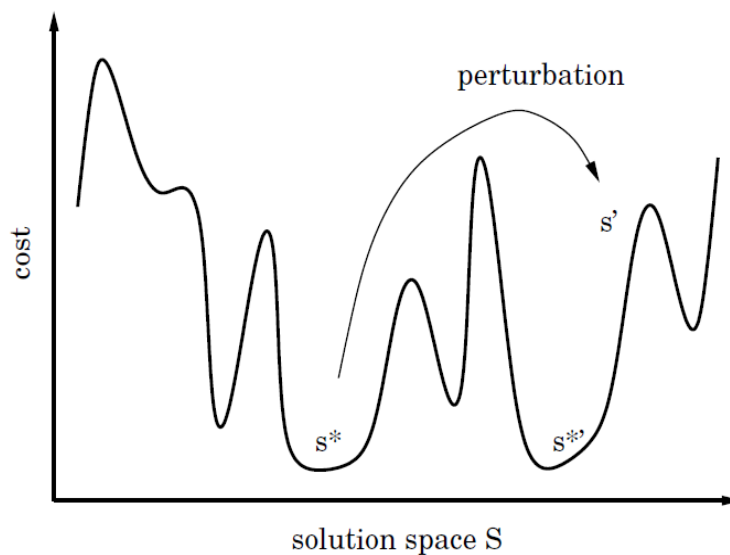


Figura 3. Representação da busca efetuada pelo ILS

Fonte: Lourenço et al. (2001)

O critério de aceitação determina se a solução $s^{*'}$ (linha 5) será aceita como nova solução corrente. O critério pode ser definido de várias maneiras. Por um lado, o critério de aceitação pode aceitar a solução obtida na etapa da busca local somente se o custo dela for melhor que o custo da solução corrente s^* , dando preferência à intensificação. Em outro extremo, o critério de aceitação pode sempre aceitar a solução $s^{*'}$, favorecendo a diversificação. Outra possibilidade, segundo LOURENÇO et al. (2001), é variar o critério de aceitação durante a execução, com uma ideia similar ao *Simulated Annealing*. A solução $s^{*'}$ é aceita se for melhor que a solução s^* . Caso contrário ela é aceita com uma probabilidade $e^{-[f(s^*) - f(s^{*'})] / T}$, onde T é a variável que representa a temperatura, decaindo durante a execução do algoritmo. Conforme a meta-

heurística vai sendo executada a probabilidade de aceitação de soluções s^* piores que s^* vai sendo reduzida, aumentando a intensificação e reduzindo a diversificação.

O parâmetro *histórico* (linhas 4 e 6) pode ser utilizado para armazenar informações acumuladas ao longo das iterações, influenciando o critério de aceitação. Segundo LOURENÇO et al. (2001) uma possibilidade seria armazenar o melhor ótimo local encontrado e, se a meta-heurística não encontrar nenhum ótimo local melhor por x iterações, o critério de aceitação retornará uma nova solução s . Esta solução pode ser criada de forma aleatória, através de uma heurística gulosa randômica ou outra modificação mais aprimorada que possibilite atingir outras regiões do espaço de busca.

3.2 ILS aplicada ao problema CMS

Nesta seção, apresentam-se as escolhas específicas dos componentes do método ILS para o problema CMS. O algoritmo proposto, resultante das escolhas relacionadas abaixo, é chamado de ILS_CMS.

- Solução inicial. Dois métodos de geração de solução inicial foram considerados: solução inicial aleatória e solução inicial construída por um algoritmo guloso, que será apresentado na Seção 3.3;
- Busca Local. A busca local implementada utiliza uma relação de vizinhança baseada na movimentação de um módulo de um cluster para outro e estratégia de seleção do melhor vizinho. O algoritmo é o mesmo apresentado na Seção 3.4, porém sem a estratégia de múltiplos reinícios aleatórios;
- Método de perturbação. Foram propostas cinco estratégias para o método de perturbação: movimentação, troca, divisão, união e explosão. A estratégia *movimentação* seleciona um módulo de forma aleatória e realiza o movimento de retirá-lo do cluster corrente e inseri-lo em outro cluster, também selecionado de forma aleatória. A estratégia *troca* seleciona, de forma aleatória, dois módulos de clusters distintos e troca seus respectivos clusters. A estratégia *divisão* realiza a divisão de um cluster selecionado em dois clusters de tamanho equivalente. A estratégia *união* realiza a união de dois clusters, selecionados de forma aleatória, em um único cluster. A estratégia *explosão* realiza a divisão de um cluster, selecionado aleatoriamente, em clusters com somente um módulo. As estratégias de

movimentação, *divisão* e *explosão* foram utilizadas em trabalhos anteriores (SEMAAN et al., 2011).

O algoritmo recebe como parâmetro um percentual a ser aplicado no método de perturbação. O percentual é calculado sobre o número de módulos da instância e mantido fixo durante toda a execução do algoritmo. Por exemplo, para uma instância com 100 módulos e percentual de 10%, a estratégia *movimentação* efetuará 10 movimentos de módulos e a estratégia *união* efetuará 10 uniões de dois clusters.

O método de perturbação do ILS_CMS pode adotar uma, duas, três, quatro ou as cinco estratégias simultaneamente. A composição das estratégias de perturbação pode ser realizada de duas maneiras: sequencial e aleatória. Tanto as estratégias de perturbação como a forma de composição são parâmetros do algoritmo. Na composição sequencial, o método de perturbação aplica as estratégias uma após a outra - primeiro a movimentação de módulos, em seguida a troca de módulos, e assim sucessivamente. Na composição aleatória, para a perturbação aplicada na solução corrente é selecionada uma das estratégias adotadas na configuração. Por exemplo, se na configuração do ILS_CMS for adotada a composição aleatória das estratégias de *movimentação* e *troca*, o método de perturbação escolherá aleatoriamente entre uma destas duas estratégias cada vez que for aplicado. Uma vez definida a configuração da forma de composição, estratégias de perturbação e percentual de perturbação, o método de perturbação permanece inalterado durante a execução do algoritmo;

- Critério de aceitação. O ILS proposto para o problema CMS utiliza como critério de aceitação aceitar a solução obtida na busca local como nova solução corrente somente se o valor de MQ for maior que o valor de MQ da solução corrente;
- Histórico da busca local. O uso de histórico consiste em verificar se ocorre melhoria na solução após a busca local por até cinco perturbações. Caso contrário, uma nova busca ocorre a partir de uma solução aleatória. Foram avaliadas configurações com e sem uso de histórico.

3.3 Algoritmo Guloso

O algoritmo utilizado como alternativa para a construção da solução inicial do ILS_CMS usa uma abordagem construtiva, na qual a solução vai sendo construída a

cada iteração utilizando um Algoritmo Guloso (CORMAK, 1971, GORDON, 1987). Os Algoritmos Gulosos consideram, na iteração corrente, a melhor decisão a ser tomada. A cada passo é selecionado um elemento da solução que ainda não foi utilizado, minimizando o incremento no custo da solução parcial, sem torná-la inviável. O Algoritmo Guloso sempre encontra a mesma solução para o problema considerado. Diferentemente, Algoritmos Gulosos Randômicos consideram escolhas aleatórias o que permite diversificar a solução final.

O algoritmo guloso proposto é baseado em abordagem aplicada na clusterização de grandes redes (CLAUSET et al., 2004) e usada para identificar as “*community structures*” (clusters), compostas pelos vértices mais densamente conectados entre si por arestas do que conectados a vértices de outros clusters. O problema de identificação de “*community structures*” pode ser representado genericamente como a identificação de grupos em grandes redes, tais como a Internet, redes sociais, redes de citações de artigos, entre outros. É um algoritmo de clusterização do tipo aglomerativo, rápido o suficiente para o particionamento hierárquico de grafos com vários milhares de vértices.

O primeiro passo do algoritmo é criar uma solução com um módulo por cluster. Por exemplo, para um problema com n módulos, a solução inicial tem n clusters distintos, contendo um módulo cada. Em seguida, o algoritmo verifica todas as combinações de dois clusters, selecionando a união dos dois clusters que produz o maior MQ . Este último passo é repetido até o algoritmo não encontrar nenhuma união de dois clusters que resulte em melhoria do valor da função MQ .

O cálculo do valor da função MQ da solução inicial com n clusters foi implementado conforme o cálculo do “Turbo MQ ” apresentado por MITCHELL (2002). O passo do algoritmo que realiza a união de dois clusters atualiza o valor corrente do MQ de um valor “delta”, chamado no trabalho de Mitchell de *Incremental Turbo MQ* (*ITurboMQ*). Quando executando um movimento de um módulo de um cluster para outro, apenas é necessário calcular os novos valores de CF dos dois clusters envolvidos no movimento e atualizar o valor do MQ corrente diminuindo os valores de CF anteriores dos dois clusters e somando os novos CF dos mesmos clusters. Esta diferença evita recalculando o CF de todos os clusters, que é uma operação mais custosa e desnecessária. No algoritmo guloso os valores de CF dos dois clusters envolvidos na união são recalculados e o valor da função MQ corrente atualizado, enquanto no algoritmo de busca local os dois CF dos clusters envolvidos na movimentação do

módulo são atualizados a partir do delta obtido da movimentação do módulo selecionado, não precisando ser recalculados.

3.4 Algoritmo de Busca Local Subida de Encosta com Reinícios Aleatórios

Algoritmos de busca local são algoritmos que exploram o espaço de soluções possíveis movendo, a cada iteração, de uma solução corrente para uma solução vizinha na região do espaço de soluções. A busca é guiada por uma função de avaliação das soluções. As soluções vizinhas são estabelecidas a partir de uma relação de vizinhança entre as soluções. O estabelecimento da relação de vizinhança é importante para que as soluções vizinhas sejam diversas o suficiente de forma que a busca não fique limitada a uma região restrita a um ótimo local ruim. Ao mesmo tempo, para que a identificação do próximo movimento e consequente percurso na vizinhança seja eficiente, a vizinhança não deve ser extensa demais (KLEINBERG e TARDOS, 2005).

Em relação à movimentação pelo espaço de soluções, o algoritmo de busca local pode utilizar diferentes estratégias. Na estratégia de seleção por primeira melhoria (*first-improvement*) a movimentação ocorre para a primeira melhor solução vizinha, enquanto na estratégia de seleção do melhor vizinho (*best-improvement*) avalia todas as soluções da vizinhança e se movimenta, na próxima iteração, para a solução melhor avaliada.

A busca local implementada no presente trabalho utiliza a estratégia subida de encosta com múltiplos reinícios aleatórios (*Random Restart Hill Climbing*) e a estratégia de seleção pelo melhor vizinho (*best-improvement*), conforme apresentadas por MANCORIDIS et al. (1998) e implementadas na ferramenta Bunch.

O algoritmo parte de uma solução inicial aleatória construída da seguinte forma: para cada módulo é gerado aleatoriamente um número entre 1 e $n/2$, onde n representa o número de módulos. Em seguida, atribui-se cada módulo ao cluster identificado pelo número aleatório correspondente. A relação de vizinhança corresponde à movimentação de um módulo de um cluster para outro. Dada uma solução com n módulos e k clusters, existem $n * (k - 1)$ possibilidades de movimento. A estratégia de análise da vizinhança utilizada calcula o valor da função MQ de todas as soluções vizinhas da solução corrente. O MQ de cada vizinho é calculado a partir do MQ da solução corrente, calculando-se a diferença nos CF (*Cluster Factor*, ver Seção 2.1) envolvidos na movimentação do módulo: uma vez que somente os CF dos clusters de origem e destino da movimentação do módulo são modificados, não há necessidade de recálculo dos CF

dos demais clusters. Com isto, o processo de cálculo da busca local é acelerado se comparado ao processo que calcula o valor da função MQ avaliando todos os CF . A estratégia de geração da solução aleatória cria soluções com no máximo $n/2$ clusters. No decorrer da busca local o número de clusters pode ser reduzido através da movimentação de um módulo de um cluster para outro.

A execução de uma busca local termina quando a busca atinge um ótimo local, nem sempre de boa qualidade. Com o objetivo de evitar o término prematuro da busca, é possível incluir um componente aleatório para fazer a busca reiniciar, por exemplo, a partir de uma nova solução aleatória, quando se atinge um ótimo local. Esta estratégia de reinício foi implementada considerando um número máximo de avaliações da solução como critério de parada. Para o valor do número máximo de avaliações foi escolhido o mesmo valor utilizado em trabalho anterior (BARROS, 2012), que é igual a 200 vezes o quadrado do número de módulos da instância.

3.5 Algoritmos Genéticos

Algoritmos Genéticos são algoritmos de busca heurística que utilizam princípios da evolução natural (HOLLAND, 1992). Entre as técnicas inspiradas na evolução natural tem-se a mutação, a seleção e a recombinação. Estas técnicas constituem os operadores genéticos do algoritmo, implementados e configurados conforme o problema abordado. No Algoritmo Genético cada solução candidata equivale a um cromossomo. Os operadores genéticos são aplicados a um conjunto de soluções candidatas, que equivalem a uma população de cromossomos, de modo a se obter uma nova população com os cromossomos mais aptos. A aptidão de um cromossomo é a medida de qualidade da solução, obtida por uma função de avaliação da solução.

A recombinação (ou *crossover*) representa a combinação biológica dos cromossomos, onde novos cromossomos são gerados a partir da combinação de genes de dois cromossomos reprodutores distintos. A mutação é um mecanismo com o objetivo de introduzir diversidade na população, evitando a convergência do algoritmo em um mínimo ou máximo local. A mutação é realizada através da perturbação de um ou mais genes de um dado cromossomo, gerando novos indivíduos.

O processo de geração de uma nova população através de operadores genéticos é repetido até que uma condição de parada seja satisfeita. A condição de parada é definida de acordo com o problema e o experimento; pode ser um número máximo de gerações,

um número máximo de avaliações de solução, um tempo de consumo de CPU, ou um valor de aptidão de solução a ser alcançado.

Conforme apresentado no Capítulo 2, a literatura apresenta diversos trabalhos comparativos envolvendo o uso de algoritmos genéticos mono-objetivo para encontrar boas soluções para o problema SMC (PRADITWONG, 2011, HARMAN et al., 2002). Nesta seção, são apresentadas duas abordagens distintas para a representação do cromossomo utilizado nos algoritmos genéticos. Estas representações serão aplicadas ao problema CMS de modo a comparar a eficiência e a eficácia do ILS_CMS proposto com os algoritmos genéticos.

A representação do cromossomo é uma questão importante no projeto de algoritmos genéticos. Uma vez que o espaço de busca constitui o conjunto de todas as soluções, cada solução deve ter pelo menos um cromossomo correspondente. Entre os princípios para construir representações eficazes (RADCLIFFE, 1991) destaca-se o princípio da Mínima Redundância, onde cada solução deve ser representada por tão poucos cromossomos quanto possível (idealmente um único cromossomo), de modo a reduzir o espaço de busca a ser percorrido pelo algoritmo genético. Embora existam técnicas (FALKENAUER, 1996, TUCKER et. al, 2005) para tratar problemas de agrupamento como o CMS, uma representação de cromossomo para essa categoria de problemas que não possua redundância ainda é uma questão de pesquisa em aberto (LEWIS e PULLIN, 2011).

As duas representações de cromossomos adotadas foram: GNE (*Grouping Number Encoding*) e GGA (*Grouping Genetic Algorithm*) (Seção 2.1). Os algoritmos genéticos utilizados para resolver heurísticamente o CMS, baseados na representação GNE e GGA, serão denominados neste trabalho de AG_GNE e AG_GGA.

O algoritmo genético AG_GNE implementado possibilita utilizar como método de seleção de cromossomos para reprodução as estratégias de *seleção por torneio* e de *seleção por classes*. Na seleção por torneio um grupo de x indivíduos é aleatoriamente escolhido e o indivíduo com melhor aptidão é selecionado. Neste trabalho foi utilizado o torneio binário, ou seja, $x = 2$. Este método de seleção foi utilizado em outros trabalhos envolvendo algoritmos genéticos aplicados ao problema CMS (BARROS, 2012, PRADITWONG, 2011). Na seleção por classes a população é dividida em três grupos: o grupo A, que representa a elite, é formado pelos $x\%$ melhores indivíduos; o grupo C, que representa o descarte, é formado pelos $y\%$ piores indivíduos, e o grupo B é formado pelos $(100-x-y)\%$ melhores indivíduos restantes. Os indivíduos do grupo C não

participam da seleção. Portanto, na seleção por classes os dois cromossomos reprodutores são selecionados dos grupos A e B, respectivamente.

O algoritmo genético AG_GNE utiliza os seguintes operadores de recombinação: *one-point crossover* e *two-point crossover*. O percentual para realização da recombinação foi mantido constante durante a execução do algoritmo, sendo igual a 80% para sistemas com menos de 100 módulos e 100% para sistemas maiores. Estes valores são compatíveis com os valores utilizados em trabalhos anteriores realizados na área (MANCORIDIS et al., 2006, PRADITWONG et al., 2011, BARROS, 2012).

Em relação ao operador de mutação, o algoritmo genético AG_GNE utilizou a *mutação uniforme*, na qual o percentual utilizado para aplicar a mutação em cada gene (módulo) do cromossomo é mantido constante ao longo das gerações, sendo utilizado o percentual de 0.4% vezes $\log n$, onde n é o número de módulos.

O algoritmo genético AG_GGA utiliza a recombinação por agrupamentos, conforme proposta de PRADITWONG (2011). Também foram introduzidos dois tipos de mutação: i) baseada em uniões de clusters, e ii) escolha aleatória entre três opções: união de dois clusters, divisão de um cluster em dois e movimentação de um módulo. O tipo de mutação a ser realizado é um parâmetro fornecido para o algoritmo, podendo ser selecionada nenhuma mutação, mutação por união de clusters ou a mutação por escolha aleatória entre três estratégias de modificação. Outra adaptação introduzida na recombinação por agrupamentos foi uma restrição na quantidade máxima de clusters a participarem na recombinação. A restrição é fornecida em forma de percentual, podendo variar entre 1% e 100% (sem restrição).

A estratégia de unir os clusters com somente um módulo, presente no estudo de PRADITWONG (2011), foi mantida. Outra estratégia adotada, com o objetivo de introduzir diversidade na população foi verificar se na nova população existiam indivíduos com o mesmo *fitness* dos indivíduos gerados pela recombinação e mutação. Se a quantidade de indivíduos com mesmo *fitness* atingir um percentual pré-definido (1% de indivíduos, calculado sobre o número de módulos da instância), é realizada uma mutação nos novos indivíduos baseada na movimentação de módulos (5% de módulos movimentados aleatoriamente).

O tamanho da população foi mantido fixo durante a execução dos algoritmos AG_GNE e AG_GGA. Foi utilizado o tamanho de 10 vezes n , onde n é o número de módulos. A população inicial é preenchida por soluções geradas aleatoriamente,

utilizando o processo de construção de soluções descrito no algoritmo de busca local (Seção 3.4).

O critério de parada utilizado é um número máximo de avaliações de soluções, utilizando-se aqui o mesmo limite estabelecido para a busca local. As taxas de mutação, recombinação, tamanho da população são as mesmas utilizadas em outros estudos envolvendo a aplicação de algoritmos genéticos para o problema CMS, tais como BARROS (2012) e PRADITWONG et al. (2011). O critério de parada é o mesmo utilizado no estudo de BARROS (2012).

3.6 Considerações finais

Este capítulo apresentou a heurística ILS_CMS como proposta de solução para o problema de Clusterização de Módulos de Software. Além da heurística ILS_CMS, foram detalhadas as implementações de outras heurísticas utilizadas na literatura, tais como uma busca local e dois algoritmos genéticos, para efeito de comparação com a heurística ILS_CMS. Foram implementados operadores genéticos utilizados em trabalhos anteriores, que se mostraram promissores, tais como um operador de recombinação baseado em agrupamentos (PRADITWONG, 2011). Para o algoritmo genético AG_GGA também foram considerados operadores genéticos que não constam na revisão bibliográfica para o problema CMS, tais como operadores de mutação baseados em agrupamentos (FALKENAUER, 1996).

O próximo capítulo descreverá os experimentos computacionais efetuados para avaliar a proposta ILS_CMS, com os resultados obtidos e ameaças à validade do estudo.

Capítulo 4 - Experimentos Computacionais

Com o objetivo de avaliar o algoritmo proposto ILS_CMS quatro estudos experimentais foram realizados. O objetivo do primeiro estudo foi avaliar o comportamento do ILS_CMS através da aplicação das estratégias apresentadas na Seção 3.2 e obter, dentre as variantes propostas, a melhor configuração do algoritmo. O segundo estudo avaliou diferentes configurações das duas abordagens baseadas em algoritmos genéticos apresentadas na Seção 3.5 visando selecionar a melhor configuração entre elas. O terceiro estudo avaliou o ILS_CMS em relação à qualidade de solução e tempo de processamento, comparando a melhor configuração do algoritmo (encontrada no primeiro estudo) com a busca local (apresentada na Seção 3.4) e a melhor configuração do algoritmo genético (encontrada no segundo estudo). O último estudo comparou os valores de MQ obtidos pelo ILS_CMS e ILS_CMS* em relação a valores prévios obtidos na literatura, para a busca local e algoritmos genéticos. Todos os experimentos foram realizados em um computador com processador Intel Core i7-2600 CPU 3.40 GHz, com memória 4GB e dedicação exclusiva. Todos os algoritmos foram codificados na linguagem Java 7 e compilados utilizando-se a JDK 1.7.0-b147.

4.1 Instâncias de Teste

As instâncias de teste foram selecionadas em sua maioria de aplicações reais *open-source* ou *free-software* e com código-fonte disponível em Java. Entre as instâncias também consta uma aplicação desenvolvida por uma empresa brasileira fora do modelo *open-source* (SEEMP). Utilizou-se a ferramenta PF-CDA ("<http://www.dependency-analyzer.org/>"), que efetua a leitura do código-fonte das aplicações e gera as instâncias no formato de arquivo "*odem*". Este formato identifica as dependências estáticas existentes entre classes da aplicação e entre classes e pacotes. De acordo com a quantidade de módulos, as instâncias foram classificadas em três categorias, conforme mostrado na Tabela 1.

Tabela 1 - Instâncias por quantidade de módulos.

Categoria	Número de Módulos	Número de instâncias
Pequeno porte	Até 100 módulos	21
Médio porte	Até 200 módulos	12
Grande porte	Acima de 200 módulos	13

Foram selecionadas 46 instâncias de diferentes portes, relacionadas na Tabela 2. A primeira coluna exibe um identificador para a instância; a segunda coluna exibe o nome do software que gerou a respectiva instância; a terceira coluna mostra o número de módulos da instância e a última coluna exibe o número de dependências entre seus módulos. A maior instância possui 483 módulos e mais de 7.000 dependências.

Tabela 2 - Conjunto de 46 instâncias para testes.

Identificador	Descrição	Módulos	Dependências
JSTL	Biblioteca padrão de Tags JSP	18	20
MTUNIS **	Sistema operacional escrito em Turing com propósitos educacionais	20	57
ISPELL **	Software para correção de erros de pronúncia e digitação	24	103
JNANOXML *	Parser de XML para Java	25	64
JODAMONEY	Biblioteca de gerenciamento monetário	26	102
JXLSREADER	Biblioteca para leitura de arquivos Excel	27	73
RCS **	Sistema para gerenciar múltiplas revisões de arquivos	29	163
SEEMP	Sistema de pequeno porte desenvolvido por empresa brasileira	31	61
APACHE *	Utilitário para compressão de arquivo	36	86
BISON **	Interpretador que converte uma descrição gramatical para C	37	179
UDTJAVA	Implementação nativa para protocolo UDT v0.5	56	227
JAVAOCR	Biblioteca para reconhecimento óptico de caracteres	59	155
SERVLET	API para Java Servlets	63	131
PFCDA_BASE	Modelo de classes de aplicativo que analisa código fonte v1.1.1	67	197
FORMS	Biblioteca para gerenciamento de GUI de formulários v1.3.0	68	270
JSCATTERPLOT *	Biblioteca para gráficos de dispersão (parte do JTreeview) v1.1.6	74	232
JFLUID	Ferramenta para <i>profiling</i> de aplicações Java, v1.7.0	82	315
JXLSCORE	Biblioteca para ler e gerar arquivos excel a partir de templates	83	330
GRAPPA **	Software para análise de rearranjo de genomas	86	295
JPASSWORD	Aplicativo para gerenciamento de senhas	96	361
JUNIT *	Biblioteca para testes unitários	100	276
XMLDOM	Biblioteca para manipulação de arquivos xml	119	209
TINYTIM *	Biblioteca em java para leitura de mapas de conhecimento	134	564
JKARYOSCOPE	Biblioteca para gerar gráficos <i>karyoscope</i> (parte do Jtreeview) v1.1.6	136	460
GAE_PLUGIN*	Google Plugin para Eclipse	140	375
JAVACC	Gerador de analisador sintático aberto	154	722
JAVAGEOM	Biblioteca para desenvolvimento de aplicativos de geometria v0.11	172	1445
INCL **	Utilitário para desenho gráfico	174	360

JDENODOGRAM *	Biblioteca para dendogramas (parte do JTreeview) v1.1.6	177	583
XMLAPI	Java XML API	184	413
JMETAL	Framework para otimização multiobjetivo com metaheurísticas	190	1137
DOM4J	Biblioteca para trabalhar com XML, XPath e XSLT	195	930
PDF_RENDERER *	Renderizador Java para arquivos PDF v0.2.1	199	629
JUNG_GRAPH_MODEL	Parte da biblioteca Jung Graph para modelagem em forma de grafo v2.0.1	207	603
JCONSOLE	Java Console (parte do JDK) v1.7.0	220	859
JUNG_VISUALIZATION *	Parte da biblioteca Jung Graph para visualização em forma de grafo v2.0.1	221	919
PFCDA_SWING *	Classes de interface de aplicativo que analisa código fonte v1.1.1	252	885
JPASSWORD_FULL	Aplicativo para gerenciamento de senhas (completo) v0.5	269	1348
JML *	Biblioteca Java para MSN Messenger v1.0	270	1745
NOTEPAD_FULL *	Caderno digital de anotações para tablets v0.2.1	299	1349
POORMANS CMS	CMS que gera páginas html estáticas	304	1118
LOG4J	Biblioteca relacionada ao controle de log de aplicações Java v1.2	308	1078
JTREEVIEW	Biblioteca que permite vários tipos de visualizações v1.1.6	329	1057
JACE	Emulador de computadores Apple	340	1524
JAWAWS	Java Web Start - parte do JAVA JRE v7	378	1403
RES_COBOL	RES - tradutor de programas em Cobol para Java	483	7163

Deste conjunto selecionou-se um subgrupo (instâncias com asterisco ao lado do seu identificador na Tabela 2) composto por quatro instâncias de pequeno porte, quatro de médio porte e quatro com mais de 200 módulos. Este grupo foi utilizado no primeiro e segundo estudos experimentais. O terceiro estudo utilizou todas as instâncias de teste. Algumas das instâncias foram utilizadas em outras publicações que abordaram o problema CMS. Por exemplo, BARROS (2012) utilizou 14 das instâncias acima em estudo envolvendo algoritmos genéticos multiobjetivo. Seis instâncias não ponderadas foram utilizadas em estudos prévios envolvendo tanto algoritmos genéticos mono-objetivo quanto multiobjetivo (PRADITWONG, 2011, PRADITWONG et al., 2011, MANCORIDIS, 2002). Estas últimas seis instâncias estão marcadas com dois asteriscos ao lado do seu identificador na Tabela 2, e participaram do quarto estudo comparativo. O quarto estudo tem como objetivo reduzir ameaças de validade ao comparar os resultados obtidos com outros da literatura, e avaliar a heurística proposta para o critério de parada de $2000 \cdot n^2$ avaliações.

4.2 Parâmetros dos Experimentos Computacionais

Uma vez que as heurísticas contêm componentes aleatórios, podendo retornar diferentes soluções a cada execução, cada algoritmo foi executado 30 vezes para cada instância, obtendo-se o valor de MQ e o tempo de execução ao final de cada rodada. A partir dos valores de MQ para cada instância, os algoritmos foram comparados em termos de eficácia das soluções (maior valor de MQ representa maior eficácia). A partir do tempo de execução obtido em cada rodada, os algoritmos foram comparados em termos de eficiência (menor tempo de execução representa maior eficiência).

Testes estatísticos foram aplicados com o objetivo de determinar se os resultados obtidos pelas heurísticas podem ser considerados significativamente distintos entre si e se o valor médio dos resultados obtidos por uma heurística é superior aos valores médios obtidos por outras heurísticas. O teste de inferência estatística não paramétrico de Wilcoxon-Mann-Whitney (FELTOVICH, 2003) foi utilizado para comparar as médias de duas amostras independentes. Este teste não requer normalidade e homocedasticidade nas séries de valores comparadas, sendo aplicado com nível de significância de 95%.

Medidas de tamanho de efeito (*effect-size*) permitem identificar quantas vezes uma heurística foi melhor que outra em uma comparação par-a-par a partir dos resultados coletados do experimento. O tamanho de efeito utilizado nos estudos experimentais relatados neste capítulo foi calculado através da métrica não paramétrica \hat{A}_{12} de Vargha e Delaney (VARGHA e DELANEY, 2000).

A ferramenta R Statistical System v.2.15.2 foi utilizada para realização dos testes estatísticos e cálculo das médias, desvios padrão e máximos.

4.3 Questões de Pesquisa

A seguir, são apresentadas as questões de pesquisa abordadas na avaliação experimental realizada como parte deste trabalho.

- **QP1: Eficácia como critério de avaliação - o algoritmo ILS_CMS produz soluções com maior valor de MQ que as demais heurísticas selecionadas?**

Em relação ao critério de qualidade das soluções, medida pelo valor de MQ de cada solução, foi investigado qual das heurísticas selecionadas produz soluções de

maior qualidade. Para tanto, os algoritmos foram executados com o mesmo número máximo de avaliações de soluções (baseado no tamanho da instância) e utilizando o mesmo conjunto de instâncias. Ao final das rodadas, os valores de MQ produzidos por eles foram coletados e comparados.

- **QP2: Eficiência como critério de avaliação - o algoritmo ILS_CMS consome menos tempo de execução que as demais heurísticas selecionadas?**

Foi investigado qual é o custo computacional de cada uma das heurísticas aplicadas ao problema CMS. O custo computacional é calculado através da média do tempo de execução dos algoritmos ao longo das diversas rodadas independentes. Para viabilizar uma comparação justa entre os algoritmos, todos adotaram o mesmo critério de parada, que é um número máximo de avaliação de soluções baseado no tamanho da instância.

4.4 Estudo do comportamento do ILS_CMS

O primeiro estudo experimental teve como objetivo avaliar o comportamento do ILS_CMS sob diferentes configurações dos seus parâmetros. Para tal, foram definidas 228 configurações de parâmetros para o algoritmo. As configurações são o resultado da combinação de diferentes estratégias de geração da solução inicial, dos métodos de perturbação e do uso do histórico da busca, conforme apresentados na Seção 3.2.

Algumas das configurações do ILS_CMS que foram avaliadas neste estudo estão listadas na Tabela 3. A primeira coluna da tabela identifica a configuração, que é numerada de 1 a 228; na segunda coluna indica-se o método para a geração de solução inicial (gulosa ou aleatória); a terceira coluna indica se o histórico da busca foi considerado ou não; a quarta coluna informa se a perturbação é sequencial ou aleatória (apenas nos casos com mais de uma operação na perturbação) e as cinco colunas seguintes indicam os métodos de perturbação considerados na configuração. O Anexo I apresenta a relação de todas as configurações utilizadas neste experimento, que utilizou a amostra de 12 instâncias de teste selecionadas na Seção 4.1.

Tabela 3. Um subconjunto das configurações avaliadas para o ILS_CMS.

Configuração	Solução Inicial	Histórico	Composição Perturbação	Movimentação	Troca	União	Divisão	Explosão
ILS1	Gulosa	Não	-	Sim	-	-	-	-
ILS3	Gulosa	Não	Sequencial	Sim	Sim	Sim	-	-
ILS16	Gulosa	Não	-	-	Sim	-	-	-
ILS63	Gulosa	Não	Aleatória	Sim	Sim	-	-	-
ILS2	Gulosa	Não	Sequencial	Sim	Sim	-	-	-
ILS64	Gulosa	Não	Aleatória	Sim	Sim	Sim	-	-
ILS8	Gulosa	Não	Sequencial	Sim	Sim	-	-	Sim
ILS77	Gulosa	Não	Aleatória	-	Sim	Sim	-	-
ILS80	Gulosa	Não	Aleatória	-	Sim	Sim	-	Sim

Para cada uma das 12 instâncias de teste e das 228 configurações selecionadas foram realizadas 30 execuções do algoritmo ILS_CMS. Na Tabela 4 são apresentadas, por instância, a melhor média do valor da solução (MQ) entre as 30 execuções de cada configuração e o valor máximo de MQ encontrado em todas as execuções do algoritmo.

Tabela 4. Melhor média e valor máximo do MQ entre as configurações do ILS_CMS

Instância	Melhor Média (MQ)	Máximo (MQ)
JNANOXML	3,82	3,82
APACHE	5,77	5,77
JSCATTERPLOT	10,74	10,74
JUNIT	11,09	11,09
TINYTIM	12,51	12,54
GAE_PLUGIN	17,33	17,34
JDENDOGRAM	26,07	26,08
PDF_RENDERER	21,92	21,99
JUNG_VISUALIZATION	20,91	21,19
PFCDA_SWING	28,99	29,08
JML	17,41	17,54
NOTEPAD_FULL	29,49	29,63

Para cada configuração foi calculado o número de instâncias em que ela produziu a melhor média da instância. Na Tabela 5 são listadas as configurações que atingiram o maior número de vezes as melhores médias (entre 4 e 7 instâncias). A tabela segue o mesmo formato da Tabela 3, adicionando uma coluna para informar o número de vezes em que a configuração atingiu a melhor média.

Tabela 5. Configurações que atingiram a melhor média em 4 ou mais instâncias

Configuração	Solução Inicial	Histórico	Composição Perturbação	Movimentação	Troca	União	Divisão	Explosão	Vitórias
ILS1	Gulosa	Não	-	Sim	-	-	-	-	7
ILS3	Gulosa	Não	Sequencial	Sim	Sim	Sim	-	-	6
ILS16	Gulosa	Não	-	-	Sim	-	-	-	6
ILS63	Gulosa	Não	Aleatória	Sim	Sim	-	-	-	6
ILS2	Gulosa	Não	Sequencial	Sim	Sim	-	-	-	5
ILS64	Gulosa	Não	Aleatória	Sim	Sim	Sim	-	-	5
ILS8	Gulosa	Não	Sequencial	Sim	Sim	-	-	Sim	4
ILS77	Gulosa	Não	Aleatória	-	Sim	Sim	-	-	4
ILS80	Gulosa	Não	Aleatória	-	Sim	Sim	-	Sim	4

A Tabela 6 lista o número de configurações que atingiram um determinado número de vezes a melhor média, considerando que a configuração que atingiu esta média com maior frequência a atingiu em sete instâncias.

Tabela 6. Quantidade de configurações que atingiram as melhores médias.

Número de vitórias	7	6	5	4	3	2	1	0
Número de configurações	1	3	2	3	11	107	40	61

Considerando-se o percentual idêntico de perturbação em cada uma das estratégias e que o método de perturbação escolhido permaneceu fixo durante cada execução do algoritmo ILS_CMS, as configurações com melhores médias indicaram que as estratégias de movimentação e troca de módulos foram mais eficazes no que diz respeito à estratégia de perturbação. Em todas as configurações vencedoras pelo menos uma destas estratégias esteve presente. A Tabela 7 mostra a média e o desvio padrão do valor de MQ retornado para cada uma das instâncias para cada configuração da Tabela 5. Os valores são apresentados em negrito quando atingem as melhores médias obtidas entre as 228 configurações.

As nove configurações com melhores médias foram comparadas aplicando-se o teste não paramétrico Wilcoxon-Mann-Whitney. A Tabela 8 apresenta os resultados comparativos dos testes estatísticos entre as quatro configurações que obtiveram as melhores médias de MQ (IL1, ILS3, ILS16 e ILS63). Os *p-values* apresentados em branco na tabela correspondem às amostras que retornaram resultados idênticos. Os valores de tamanho de efeito maiores que 50% são apresentados em negrito. O Anexo II contém os testes estatísticos completos entre as nove configurações.

Tabela 7. Média e desvio padrão por instância das melhores configurações analisadas para o ILS_CMS

INSTÂNCIAS	ILS 1	ILS 3	ILS 16	ILS 63	ILS 2	ILS 64	ILS 8	ILS 77	ILS 80
JNANOXML	3,82 ± 0,00	3,82 ± 0,00	3,82 ± 0,00	3,82 ± 0,00	3,80 ± 0,00	3,82 ± 0,00	3,82 ± 0,00	3,82 ± 0,00	3,82 ± 0,00
APACHE	5,77 ± 0,00	5,77 ± 0,00	5,77 ± 0,00	5,77 ± 0,00	5,77 ± 0,00	5,76 ± 0,01	5,77 ± 0,00	5,76 ± 0,01	5,76 ± 0,01
JSCATTERPLOT	10,74 ± 0,02	10,74 ± 0,02	10,74 ± 0,00	10,74 ± 0,01	10,74 ± 0,01	10,73 ± 0,02	10,74 ± 0,02	10,73 ± 0,03	10,72 ± 0,03
JUNIT	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00	11,09 ± 0,00
TINYTIM	12,51 ± 0,02	12,47 ± 0,02	12,51 ± 0,02	12,5 ± 0,02	12,49 ± 0,02	12,51 ± 0,02	12,47 ± 0,02	12,51 ± 0,02	12,5 ± 0,02
GAE_PLUGIN	17,28 ± 0,02	17,33 ± 0,02	17,29 ± 0,02	17,29 ± 0,02	17,28 ± 0,01	17,31 ± 0,02	17,31 ± 0,02	17,32 ± 0,02	17,32 ± 0,02
JDENDOGRAM	26,07 ± 0,01	26,07 ± 0,01	26,07 ± 0,01	26,07 ± 0,01	26,07 ± 0,01	26,07 ± 0,01	26,06 ± 0,01	26,06 ± 0,01	26,06 ± 0,01
PDF_RENDERER	21,85 ± 0,14	21,72 ± 0,14	21,78 ± 0,2	21,82 ± 0,11	21,78 ± 0,16	21,86 ± 0,12	21,75 ± 0,17	21,88 ± 0,15	21,92 ± 0,10
JUNG_VISUALIZATION	20,88 ± 0,21	20,79 ± 0,2	20,76 ± 0,16	20,81 ± 0,21	20,84 ± 0,17	20,82 ± 0,17	20,78 ± 0,19	20,77 ± 0,2	20,72 ± 0,21
PFCDA_SWING	28,99 ± 0,03	28,94 ± 0,02	28,97 ± 0,03	28,98 ± 0,03	28,95 ± 0,02	28,97 ± 0,03	28,94 ± 0,02	28,97 ± 0,03	28,96 ± 0,03
JML	17,40 ± 0,05	17,36 ± 0,03	17,39 ± 0,04	17,4 ± 0,03	17,38 ± 0,04	17,41 ± 0,04	17,37 ± 0,04	17,41 ± 0,05	17,41 ± 0,04
NOTEPAD_FULL	29,48 ± 0,05	29,41 ± 0,06	29,48 ± 0,05	29,49 ± 0,06	29,41 ± 0,05	29,47 ± 0,05	29,41 ± 0,05	29,46 ± 0,06	29,45 ± 0,08

Tabela 8. P-value e tamanho de efeito na comparação entre as 4 melhores configurações do ILS_CMS

INSTÂNCIAS	ILS1 > ILS3		ILS1 > ILS16		ILS63 > ILS3		ILS63 > ILS16		ILS1 > ILS63	
	P-value	Effect-Size	P-value	Effect-Size	P-value	Effect-Size	P-value	Effect-Size	P-value	Effect-Size
JNANOXML	-	50%	-	50%	-	50%	-	50%	-	50%
APACHE	1,000	50%	0,161	47%	0,161	53%	-	50%	0,161	47%
JSCATTERPLOT	0,452	53%	0,161	47%	0,371	54%	0,161	47%	0,959	50%
JUNIT	-	50%	-	50%	-	50%	-	50%	-	50%
TINYTIM	<0,001	93%	0,684	53%	<0,001	90%	0,399	44%	0,217	59%
GAE_PLUGIN	<0,001	8%	0,435	44%	<0,001	8%	0,510	45%	0,833	48%
JDENDOGRAM	0,797	52%	0,881	49%	0,884	51%	0,477	45%	0,858	51%
PDF_RENDERER	< 0,001	77%	0,208	60%	0,001	76%	0,662	53%	0,208	60%
JUNG_VISUALIZATION	0,088	63%	0,015	68%	0,719	53%	0,343	57%	0,224	59%
PFCDA_SWING	<0,0001	91%	0,004	71%	<0,001	83%	0,234	59%	0,115	62%
JML	< 0,001	79%	0,483	55%	<0,001	83%	0,181	60%	0,483	45%
NOTEPAD_FULL	<0,0001	82%	0,965	50%	<0,001	84%	0,356	57%	0,356	43%

A primeira coluna da Tabela 8 informa o nome das instâncias, que estão apresentadas em ordem crescente do seu número de módulos. A segunda e terceira colunas informam o *p-value* do teste de inferência e o tamanho de efeito para as configurações ILS1 e ILS3. Um *p-value* menor que 0,05 (em negrito) indica resultados significativamente diferentes com 95% de certeza. A comparação entre o ILS1 e ILS3 mostrou que as amostras são estatisticamente diferentes para 6 instâncias, apresentando um tamanho de efeito favorável para o ILS1 maior que 75% nas três maiores instâncias.

A quarta e quinta colunas da Tabela 8 mostram os valores de *p-value* e tamanho de efeito comparando o ILS1 e o ILS16. Somente para duas instâncias os resultados se mostraram estatisticamente diferentes, mas os valores de tamanho de efeito para o ILS1 foram superiores a partir das instâncias com mais de 199 módulos (PDF_RENDERER).

A sexta e sétima colunas da Tabela 8 mostram os valores de *p-value* e tamanho de efeito comparando o ILS63 e o ILS3. As diferenças entre as médias das amostras se mostraram estatisticamente significativas em 6 instâncias e nas três maiores instâncias o ILS63 apresentou um tamanho de efeito superior a 80%.

A oitava e nona coluna mostram os valores de *p-value* e tamanho de efeito entre o ILS63 e ILS16. Embora os resultados não tenham sido estatisticamente significativos com grau de confiança de 95%, o ILS63 mostrou um tamanho de efeito um pouco superior a partir das instâncias maiores que 199 módulos (PDF_RENDERER).

As últimas duas colunas da Tabela 8 informam os valores de *p-value* e tamanho de efeito na comparação entre o ILS1 e ILS63. Em nenhuma instância a diferença das médias de *MQ* de soluções produzidas pelo ILS1 e pelo ILS63 se mostrou estatisticamente significativa com 95% de confiança. Na análise do tamanho do efeito, considerando-se as instâncias menores que 199 módulos (pequeno porte), as configurações foram equivalentes, com resultados próximos de 50%. Para instâncias a partir de 199 módulos, a ILS1 apresentou resultados melhores em três instâncias, enquanto a ILS63 apresentou resultados melhores em duas instâncias.

Estes resultados indicam que as configurações ILS1 e ILS63 são as mais promissoras dentre as avaliadas. Embora as soluções entre estas configurações não tenham sido estatisticamente significativas com grau de confiança de 95%, a ILS1 apresentou uma média de tamanho de efeito superior (52%), sendo selecionada para participar do estudo comparativo com as demais heurísticas. As configurações ILS1 e ILS63 distinguem-se apenas em relação ao método de perturbação: enquanto a ILS1 utiliza unicamente a estratégia de *movimentação* como método de perturbação, o método de perturbação da ILS63 seleciona aleatoriamente entre as estratégias de *movimentação e troca*.

Outra análise realizada a partir dos resultados do estudo foi a influência da estratégia para geração da solução inicial na capacidade do algoritmo encontrar boas soluções. Cada configuração do ILS_CMS usando a estratégia de geração de solução inicial pelo algoritmo guloso foi comparada com a configuração equivalente utilizando a estratégia de geração de solução inicial aleatória e mantendo os demais parâmetros

com os mesmos valores. Esta estratégia de análise resultou em 114 comparações para cada instância. Cada comparação foi realizada com base na média dos valores de MQ obtidos em 30 rodadas independentes para cada instância.

Tabela 9. Comparação entre as estratégias de geração da solução inicial

Instância	ILS_CMS com Gerador Inicial Guloso	ILS_CMS com Gerador Inicial Aleatório
JNANOXML	64	0
APACHE	27	0
JSCATTERPLOT	32	0
JUNIT	100	0
TINYTIM	114	0
GAE_PLUGIN	3	18
JDENDOGRAM	96	0
PDF_RENDERER	65	7
JUNG_VISUALIZATION	1	35
PFCDA_SWING	114	0
JML	105	0
NOTEPAD_FULL	114	0
TOTAL DE VITÓRIAS	835	60
PERCENTUAL DE VITÓRIAS	93,30%	6,70%

Na Tabela 9 é exibido, para cada instância, o total de configurações vencedoras para cada estratégia de geração da solução inicial. Na segunda coluna, uma vitória é contabilizada quando uma configuração utilizando a solução inicial gulosa obtém melhor média em uma dada instância que a configuração equivalente utilizando a solução inicial aleatória. Na terceira coluna, uma vitória é contabilizada quando uma configuração utilizando a solução inicial aleatória obtém melhor média que a configuração equivalente utilizando a solução inicial gulosa. As vitórias são contabilizadas apenas quando existe uma diferença estatisticamente significativa entre as duas configurações (com base no teste de inferência com 95% de certeza).

As comparações entre configurações equivalentes indicaram que a estratégia gulosa foi mais eficaz que a estratégia aleatória na geração da solução inicial. Foram realizadas 1368 comparações (114 comparações por instância). Destas, 895 foram estatisticamente significativas (65,42% do total). A estratégia gulosa foi vencedora em 93% das comparações. A preponderância da estratégia gulosa é mais evidente nas instâncias com maior número de módulos: para as três maiores instâncias, a estratégia gulosa foi vencedora 100% das vezes.

Além da análise realizada sobre a influência do uso da solução inicial gulosa, foi realizada uma análise similar quanto ao uso do histórico, verificando se este mecanismo possibilitou melhoria nas médias obtidas nas diferentes configurações do ILS_CMS. O uso de histórico foi introduzido no algoritmo para verificar se após a etapa de busca local ocorreu melhoria da solução. Com o uso de histórico, caso não tenha havido melhoria por cinco buscas locais consecutivas, efetuadas sobre as soluções produzidas pelo método de perturbação, a busca parte de uma nova solução aleatória em vez da obtida através do método de perturbação.

Como na análise pela estratégia de geração da solução inicial, as configurações foram divididas em dois grupos segundo o uso do histórico. Cada configuração do ILS_CMS usando a estratégia de histórico foi comparada com uma configuração equivalente sem a utilização deste recurso (e com todos os outros parâmetros idênticos), resultando em 114 comparações para cada instância. A comparação foi realizada com base no valor das médias de 30 rodadas obtidas para cada instância. Foram consideradas apenas as comparações entre médias de amostras que se mostraram estatisticamente significativas com grau de confiança de 95%.

Tabela 10. Comparativo entre as estratégias usando ou não histórico

Instância	ILS_CMS sem uso de histórico	ILS_CMS com uso de histórico
JNANOXML	2	17
APACHE	14	14
JSCATTERPLOT	43	21
JUNIT	0	15
TINYTIM	51	0
GAE_PLUGIN	3	32
JDENDOGRAM	3	0
PDF_RENDERER	12	17
JUNG_VISUALIZATION	4	6
PFCDA_SWING	10	0
JML	11	0
NOTEPAD_FULL	7	0
TOTAL DE VITÓRIAS	160	122
PERCENTUAL DE VITÓRIAS	56,74%	43,26%

O número de configurações vencedoras em cada instância é listado na Tabela 10. Do total de 1368 comparações, apenas 282 comparações se mostraram estatisticamente significativas (20,61% do total). As configurações sem histórico obtiveram vitória em 57% das comparações, enquanto as configurações com histórico obtiveram vitória em

43% das vezes. Este resultado corrobora os dados da Tabela 5, onde observamos que as melhores configurações para as instâncias de teste não usavam histórico.

4.5 Estudo do comportamento dos algoritmos genéticos

O segundo estudo experimental avaliou 14 configurações de operadores genéticos utilizando a representação GNE (AG_GNE) e seis configurações utilizando a representação GGA (AG_GGA). O objetivo deste estudo foi obter uma configuração eficiente de algoritmo genético para ser usada no estudo comparativo entre diferentes abordagens heurísticas (terceiro estudo experimental apresentado neste capítulo). Este experimento contemplou alguns dos operadores genéticos comumente usados em trabalhos dentro do tema, bem como operadores genéticos ainda não aplicados ao problema CMS, segundo o levantamento bibliográfico realizado e reportado no Capítulo 2 e a seleção de configurações apresentada na Seção 3.5. Este estudo também utilizou a amostra composta pelas 12 instâncias de teste.

As configurações utilizadas na representação GNE estão listadas na Tabela 11. A primeira coluna contém um identificador para cada configuração do algoritmo, a segunda coluna apresenta o método de recombinação (*one-point crossover* ou *two-point crossover*) e a última coluna apresenta o método de seleção (por torneio ou por classes) utilizado em cada configuração. Por exemplo, a configuração GNE1 utiliza o operador de recombinação *one-point crossover* e o método de seleção por torneio.

As configurações resultam da combinação dos operadores genéticos de recombinação, dos operadores de seleção e dos diferentes percentuais de elitismo. Junto com a seleção por torneio, foi utilizado o elitismo das duas melhores soluções e elitismo sobre os percentuais 10%, 20% e 30% das melhores soluções da população. Na seleção por classes, foram utilizados os percentuais (10%, 85%), (20%, 75%) e (30%, 65%) para representar os grupos A e B (ver Seção 3.5). O grupo C corresponde ao grupo de descarte e foi utilizado somente na seleção por classes.

Tabela 11. Configurações avaliadas para o algoritmo genético AG_GNE

Configuração	Recombinação	Seleção
GNE1	One-point crossover	Torneio (elitismo=2)
GNE2	One-point crossover	Torneio (elitismo=10%)
GNE3	One-point crossover	Por classes (A=10%, B=85%, C=5%)
GNE4	One-point crossover	Torneio (elitismo=20%)
GNE5	One-point crossover	Por classes (A=20%, B=75%, C=5%)
GNE6	One-point crossover	Torneio (elitismo=30%)
GNE7	One-point crossover	Por classes (A=30%, B=65%, C=5%)
GNE8	Two-point crossover	Torneio (elitismo=2)
GNE9	Two-point crossover	Torneio (elitismo=10%)
GNE10	Two-point crossover	Por classes (A=10%, B=85%, C=5%)
GNE11	Two-point crossover	Torneio (elitismo=20%)
GNE12	Two-point crossover	Por classes (A=20%, B=75%, C=5%)
GNE13	Two-point crossover	Torneio (elitismo=30%)
GNE14	Two-point crossover	Por classes (A=30%, B=65%, C=5%)

Além das configurações utilizando a representação GNE, foram realizados experimentos com o algoritmo genético utilizando a representação GGA e um operador de recombinação baseado em agrupamentos (PRADITWONG, 2011). Na Tabela 12 são apresentadas as configurações deste algoritmo genético. Adicionalmente à proposta de PRADITWONG (2011), que não utilizou mutação, foram utilizados dois operadores de mutação: um realizando a união de clusters e outro escolhendo aleatoriamente entre a união de clusters, a divisão de clusters e movimentação de módulos. Outra opção de configuração foi utilizar ou não um número máximo de clusters para participar em cada recombinação, informado como um percentual sobre o número de módulos da instância e tendo como objetivo impedir que a recombinação trocasse um grande número de clusters entre os dois cromossomos pai. A combinação das três opções de operador de mutação e a restrição do número máximo de clusters a participarem da recombinação resultou em seis configurações avaliadas.

Tabela 12. Configurações avaliadas para o algoritmo genético AG_GGA

Configuração	Mutação	Restrição na recombinação GGA
GGA1	-	-
GGA2	União de clusters	-
GGA3	Movimentação + União + Divisão	-
GGA4	-	10%
GGA5	União de clusters	10%
GGA6	Movimentação + União + Divisão	10%

Na Tabela 13 e na Tabela 14 estão os resultados com as médias e desvios padrão dos valores MQ para o algoritmo genético AG_GNE ao longo de 30 rodadas independentes para cada instância. A Tabela 13 apresenta os resultados colhidos para as configurações com o operador de recombinação *one-point crossover*. A Tabela 14 mostra os resultados das configurações com o operador *two-point crossover*. A melhor média de cada instância é apresentada em negrito.

Tabela 13. Resultados do AG_GNE para o operador *one-point crossover*.

INSTÂNCIAS	GENE1	GENE2	GENE3	GENE4	GENE5	GENE6	GENE7
JNANOXML	3,79 ± 0,03	3,79 ± 0,02	3,79 ± 0,03	3,79 ± 0,03	3,78 ± 0,03	3,79 ± 0,03	3,79 ± 0,03
APACHE	5,75 ± 0,04	5,74 ± 0,03	5,74 ± 0,02	5,74 ± 0,04	5,73 ± 0,04	5,74 ± 0,03	5,74 ± 0,04
JSCATTERPLOT	10,17 ± 0,14	10,67 ± 0,05	10,67 ± 0,06	10,67 ± 0,05	10,67 ± 0,05	10,66 ± 0,07	10,68 ± 0,05
JUNIT	9,50 ± 0,20	11,03 ± 0,06	11,03 ± 0,05	11,03 ± 0,06	11,05 ± 0,05	11,05 ± 0,04	11,04 ± 0,05
TINYTIM	8,65 ± 0,22	12,11 ± 0,11	12,22 ± 0,08	12,26 ± 0,08	12,19 ± 0,1	12,30 ± 0,06	12,19 ± 0,11
GAE_PLUGIN	12,08 ± 0,34	17,07 ± 0,09	17,27 ± 0,05	17,29 ± 0,04	17,2 ± 0,03	17,30 ± 0,03	17,10 ± 0,05
JDENDOGRAM	14,75 ± 0,50	24,25 ± 0,20	25,02 ± 0,18	25,16 ± 0,14	24,6 ± 0,22	25,47 ± 0,13	24,31 ± 0,19
PDF_RENDERER	12,04 ± 0,41	19,85 ± 0,32	20,52 ± 0,21	20,7 ± 0,25	20,24 ± 0,20	20,95 ± 0,18	20,08 ± 0,16
JUNG_VISUALIZATION	10,28 ± 0,35	18,39 ± 0,21	19,15 ± 0,24	19,42 ± 0,18	18,82 ± 0,22	19,87 ± 0,24	18,68 ± 0,24
PFCDA_SWING	11,85 ± 0,35	22,97 ± 0,37	23,88 ± 0,36	24,72 ± 0,29	23,48 ± 0,34	25,44 ± 0,24	22,93 ± 0,30
JML	7,62 ± 0,27	13,61 ± 0,22	14,03 ± 0,20	14,52 ± 0,21	13,75 ± 0,2	14,87 ± 0,21	13,50 ± 0,19
NOTEPAD_FULL	10,71 ± 0,39	21,8 ± 0,37	22,7 ± 0,40	23,6 ± 0,32	22,26 ± 0,4	24,43 ± 0,28	22,10 ± 0,37

Tabela 14. Resultados do AG_GNE para o operador *two-point crossover*.

INSTÂNCIAS	GENE8	GENE9	GENE10	GENE11	GENE12	GENE13	GENE14
JNANOXML	3,80 ± 0,02	3,79 ± 0,03	3,79 ± 0,02	3,78 ± 0,04	3,79 ± 0,03	3,77 ± 0,03	3,80 ± 0,03
APACHE	5,76 ± 0,02	5,74 ± 0,04	5,73 ± 0,04	5,74 ± 0,04	5,74 ± 0,02	5,74 ± 0,03	5,75 ± 0,03
JSCATTERPLOT	10,45 ± 0,12	10,66 ± 0,06	10,66 ± 0,07	10,66 ± 0,06	10,69 ± 0,06	10,68 ± 0,05	10,68 ± 0,05
JUNIT	9,80 ± 0,15	11,04 ± 0,05	11,03 ± 0,06	11,05 ± 0,04	11,06 ± 0,04	11,06 ± 0,05	11,05 ± 0,04
TINYTIM	9,22 ± 0,23	12,09 ± 0,11	12,24 ± 0,12	12,27 ± 0,09	12,23 ± 0,10	12,31 ± 0,07	12,22 ± 0,09
GAE_PLUGIN	12,99 ± 0,2	17,19 ± 0,08	17,29 ± 0,04	17,28 ± 0,03	17,27 ± 0,06	17,29 ± 0,04	17,22 ± 0,05
JDENDOGRAM	15,86 ± 0,42	24,51 ± 0,24	25,21 ± 0,19	25,36 ± 0,11	24,95 ± 0,18	25,58 ± 0,12	24,68 ± 0,22
PDF_RENDERER	13,08 ± 0,3	20,17 ± 0,27	20,79 ± 0,20	20,85 ± 0,20	20,48 ± 0,26	21,19 ± 0,19	20,41 ± 0,22
JUNG_VISUALIZATION	10,96 ± 0,3	18,83 ± 0,24	19,48 ± 0,25	19,83 ± 0,23	19,27 ± 0,21	20,16 ± 0,16	19,12 ± 0,23
PFCDA_SWING	12,86 ± 0,42	23,84 ± 0,46	24,77 ± 0,32	25,54 ± 0,27	24,49 ± 0,26	26,25 ± 0,30	24,14 ± 0,32
JML	7,93 ± 0,22	14,03 ± 0,24	14,56 ± 0,23	14,98 ± 0,20	14,31 ± 0,21	15,28 ± 0,16	14,10 ± 0,18
NOTEPAD_FULL	11,70 ± 0,39	22,73 ± 0,44	23,4 ± 0,38	24,55 ± 0,29	23,34 ± 0,38	25,47 ± 0,28	22,90 ± 0,35

A Tabela 13 e a Tabela 14 indicam que os resultados obtidos pelo operador *two-point crossover* foram superiores aos obtidos pelo *one-point crossover*. Para todas as instâncias com mais de 170 módulos (a partir da instância JDENDOGRAM), o operador *two-point crossover* foi sempre melhor que o *one-point crossover* considerando a comparação entre uma configuração e sua correspondente utilizando o mesmo operador de seleção. Para o total de 84 comparações (12 instâncias vezes 7 pares de configurações), o operador *two-point crossover* obteve melhores resultados em 81% das

comparações, o *one-point crossover* foi melhor em 11% das vezes e ocorreram 8% empates. Se forem consideradas somente as comparações onde os resultados são estatisticamente distintos entre si (Anexo III), o operador *two-point-crossover* obteve melhor resultado em 100% das comparações. A partir do estudo das médias obtidas, também é possível observar que a configuração GNE13 atingiu as melhores médias em 8 das 12 instâncias, incluindo as 6 maiores instâncias.

Considerando as configurações que atingiram as melhores médias de *MQ* em alguma instância (GNE13, GNE8, GNE12, GNE14 e GNE6) foi realizada uma análise estatística entre os resultados destas configurações. A Tabela 15 apresenta, para cada instância, os valores de *p-value* e tamanho de efeito para a comparação entre a GNE13 e as configurações GNE8, GNE12, GNE14 e GNE6. As diferenças das médias de GNE13 em relação às demais configurações se mostraram estatisticamente significativas para instâncias maiores que 177 módulos (a partir da instância JDENDOGRAM), sendo o *p-value* da comparação entre elas menor que 0,001. A GNE13 também apresentou tamanho de efeito próximo a 100% nas instâncias maiores, alcançando bons resultados também nas instâncias menores que 100 módulos (entre 31% e 60%).

Tabela 15. *P-values* e tamanho de efeito da configuração GNE13

INSTÂNCIAS	GNE13 > GNE14		GNE13 > GNE12		GNE13 > GNE8		GNE13 > GNE6		Melhor Média
	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	
JNANOXML	0,010	31%	0,153	39%	0,005	29%	0,181	40%	GNE8, 14
APACHE	0,275	42%	0,509	55%	0,005	32%	0,411	56%	GNE8
JSCATTERPLOT	0,552	55%	0,278	42%	<0,001	97%	0,214	59%	GNE12
JUNIT	0,192	60%	0,321	57%	<0,001	100%	0,200	60%	GNE12, 13
TINYTIM	<0,001	79%	0,001	75%	<0,001	100%	0,343	57%	GNE13
GAE_PLUGIN	<0,001	90%	0,052	65%	<0,001	100%	0,784	48%	GNE6
JDENDOGRAM	<0,001	100%	<0,001	100%	<0,001	100%	0,003	72%	GNE13
PDF_RENDERER	<0,001	99%	<0,001	99%	<0,001	100%	<0,001	82%	GNE13
JUNG_VISUALIZATION	<0,001	100%	<0,001	100%	<0,001	100%	<0,001	87%	GNE13
PFCDA_SWING	<0,001	100%	<0,001	100%	<0,001	100%	<0,001	99%	GNE13
JML	<0,001	100%	<0,001	100%	<0,001	100%	<0,001	93%	GNE13
NOTEPAD_FULL	<0,001	100%	<0,001	100%	<0,001	100%	<0,001	100%	GNE13

Na Tabela 16 estão os resultados com a média e desvio padrão dos valores de *MQ* para o algoritmo genético AG_GGA ao longo de 30 rodadas independentes para cada instância. As melhores médias de cada instância são apresentadas em negrito. Os resultados indicam que as configurações onde foi introduzida a restrição de número máximo de grupos a serem recombinados (GGA4, GGA5 e GGA6) foram superiores que a configuração similar sem a restrição. As configurações com operador de mutação baseado na união de clusters ou na seleção aleatória entre as três estratégias também se

mostraram superiores às versões sem operador de mutação (GGA1 e GGA4). As configurações GGA6 e GGA5 foram as que obtiveram maior número de vitórias por instância. A GGA6 atingiu as melhores médias em onze instâncias enquanto a GGA5 atingiu as melhores médias em quatro instâncias. A média do tamanho de efeito entre as configurações GGA6 e GGA5 foi de 55% a favor da GGA6, sendo esta configuração selecionada como a mais promissora. A Tabela 17 apresenta os *p-values* e tamanhos de efeito na comparação entre GGA6 e as demais configurações.

Tabela 16. Resultados para o algoritmo genético AG_GGA

INSTÂNCIAS	GGA1	GGA2	GGA3	GGA4	GGA5	GGA6
JNANOXML	3,80 ± 0,02	3,81 ± 0,01	3,80 ± 0,02	3,81 ± 0,01	3,81 ± 0,01	3,81 ± 0,01
APACHE	5,73 ± 0,05	5,75 ± 0,04	5,75 ± 0,04	5,75 ± 0,04	5,76 ± 0,03	5,77 ± 0,00
JSCATTERPLOT	10,72 ± 0,03	10,73 ± 0,02	10,73 ± 0,02	10,71 ± 0,03	10,74 ± 0,01	10,74 ± 0,00
JUNIT	11,08 ± 0,02	11,08 ± 0,02	11,08 ± 0,02	11,08 ± 0,02	11,08 ± 0,01	11,08 ± 0,01
TINYTIM	12,25 ± 0,09	12,41 ± 0,06	12,40 ± 0,08	12,22 ± 0,11	12,40 ± 0,08	12,45 ± 0,05
GAE_PLUGIN	17,29 ± 0,03	17,29 ± 0,03	17,31 ± 0,03	17,30 ± 0,03	17,30 ± 0,02	17,29 ± 0,04
JDENODOGRAM	26,00 ± 0,04	26,02 ± 0,04	26,02 ± 0,05	25,99 ± 0,05	26,02 ± 0,04	26,03 ± 0,03
PDF_RENDERER	21,58 ± 0,15	21,80 ± 0,12	21,75 ± 0,16	21,57 ± 0,17	21,82 ± 0,13	21,82 ± 0,12
JUNG_VISUALIZ	20,93 ± 0,13	21,00 ± 0,11	21,01 ± 0,12	20,92 ± 0,14	21,04 ± 0,09	21,05 ± 0,07
PFCDA_SWING	28,54 ± 0,18	28,73 ± 0,18	28,82 ± 0,09	28,60 ± 0,12	28,81 ± 0,13	28,85 ± 0,08
JML	17,04 ± 0,15	17,22 ± 0,11	17,27 ± 0,11	17,03 ± 0,16	17,29 ± 0,10	17,35 ± 0,09
NOTEPAD_FULL	29,03 ± 0,19	29,36 ± 0,11	29,42 ± 0,08	29,12 ± 0,14	29,41 ± 0,11	29,49 ± 0,10

Tabela 17. *P-values* e tamanho de efeito da configuração GGA6

INSTÂNCIAS	GGA6>GGA1		GGA6>GGA2		GGA6>GGA3		GGA6>GGA4		GGA6>GGA5	
	PV	ES	PV	ES	PV	ES	PV	ES	PV	ES
JNANOXML	0,161	59%	0,452	55%	0,180	58%	0,496	46%	0,773	48%
APACHE	<0,001	70%	0,011	60%	0,021	58%	0,011	60%	0,161	53%
JSCATTERPLOT	<0,001	70%	0,022	58%	0,003	63%	<0,001	77%	0,334	52%
JUNIT	0,121	61%	0,060	64%	0,104	62%	0,123	61%	0,907	51%
TINYTIM	<0,001	98%	0,006	71%	0,002	73%	<0,001	99%	0,009	70%
GAE_PLUGIN	1,000	50%	0,317	58%	0,211	41%	0,619	46%	0,419	44%
JDENODOGRAM	0,011	69%	0,255	59%	0,929	51%	0,003	72%	0,626	54%
PDF_RENDERER	<0,001	89%	0,515	55%	0,130	61%	<0,001	88%	0,756	48%
JUNG_VISUALIZATION	<0,001	81%	0,061	64%	0,219	59%	<0,001	81%	0,865	51%
PFCDA_SWING	<0,001	94%	0,002	73%	0,237	59%	<0,001	97%	0,286	58%
JML	<0,001	96%	<0,001	82%	0,009	69%	<0,001	96%	0,046	65%
NOTEPAD_FULL	<0,001	97%	<0,001	78%	0,031	66%	<0,001	100%	0,029	66%

4.6 Estudo comparativo entre diferentes abordagens

Este terceiro estudo experimental compara quatro abordagens heurísticas: a melhor configuração do algoritmo ILS_CMS, obtida no primeiro estudo experimental; as melhores versões dos algoritmos AG_GNE e AG_GGA, obtidas no segundo estudo

experimental; e a busca local (BL) com reinícios aleatórios. As diferentes abordagens serão comparadas em termos de eficácia (qualidade das soluções) e eficiência (tempo de execução), de modo a indicar a melhor abordagem heurística para a formulação mono-objetivo do problema CMS. Este estudo considera todas as instâncias apresentadas na Tabela 2.

4.6.1 Comparação dos valores médios de MQ

Nesta seção as abordagens estudadas são comparadas em relação à qualidade da solução que elas produzem. Foram obtidas as médias e desvios padrão de MQ para cada uma das 46 instâncias. Os resultados foram divididos de acordo com o tamanho das instâncias (até 100 módulos, entre 101 e 200 módulos e maior que 200 módulos) e são apresentados na Tabela 18, na Tabela 20 e na Tabela 22, respectivamente. A melhor média de cada instância é apresentada em negrito.

Considerando as instâncias de pequeno porte, é possível notar que o algoritmo ILS_CMS obteve melhor média em 16 das 21 instâncias, enquanto o algoritmo genético AG_GGA obteve melhor média em 12 das 21 instâncias. Quando o ILS_CMS é comparado com a busca local, os testes estatísticos demonstram que as médias são significativamente distintas entre si para 19 das 21 instâncias e o tamanho do efeito médio é de 80% a favor do ILS_CMS. O ILS_CMS também se mostrou superior ao algoritmo genético GNE, apresentando tamanho de efeito médio a seu favor de 76%. O algoritmo genético AG_GGA obteve médias de MQ significativamente superiores que a melhor configuração do algoritmo genético AG_GNE em 18 das 21 instâncias de pequeno porte. Por fim, o ILS_CMS se mostrou levemente inferior ao algoritmo genético AG_GGA nas instâncias de pequeno porte, apresentando um tamanho de efeito médio de 47%.

Tabela 18. Médias e desvios padrão do *MQ* para instâncias de pequeno porte

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)
JSTL	3,22 ± 0,08	3,31 ± 0,04	3,30 ± 0,04	3,29 ± 0,06
MTUNIS	2,28 ± 0,02	2,31 ± 0,00	2,29 ± 0,03	2,31 ± 0,00
ISPELL	2,36 ± 0,01	2,35 ± 0,01	2,34 ± 0,01	2,36 ± 0,00
JNANOXML	3,81 ± 0,02	3,82 ± 0,00	3,77 ± 0,03	3,81 ± 0,01
JODAMONEY	2,73 ± 0,02	2,75 ± 0,00	2,73 ± 0,03	2,75 ± 0,00
JXLSREADER	3,59 ± 0,02	3,60 ± 0,00	3,58 ± 0,03	3,60 ± 0,00
RCS	2,26 ± 0,01	2,24 ± 0,02	2,24 ± 0,04	2,25 ± 0,02
SEEMP	4,63 ± 0,02	4,65 ± 0,00	4,64 ± 0,02	4,65 ± 0,00
APACHE	5,74 ± 0,02	5,77 ± 0,00	5,74 ± 0,03	5,77 ± 0,00
BISON	2,66 ± 0,01	2,69 ± 0,01	2,67 ± 0,02	2,69 ± 0,01
UDTJAVA	5,23 ± 0,02	5,28 ± 0,01	5,24 ± 0,03	5,28 ± 0,01
JVAOCR	8,93 ± 0,04	9,00 ± 0,03	8,95 ± 0,04	9,02 ± 0,01
SERVLET	9,45 ± 0,05	9,47 ± 0,06	9,45 ± 0,11	9,50 ± 0,07
PFCDA_BASE	7,33 ± 0,01	7,33 ± 0,01	7,30 ± 0,03	7,32 ± 0,01
FORMS	8,22 ± 0,04	8,33 ± 0,00	8,30 ± 0,03	8,32 ± 0,01
JSCATTERPLOT	10,64 ± 0,03	10,74 ± 0,02	10,68 ± 0,05	10,74 ± 0,00
JFLUID	6,50 ± 0,04	6,58 ± 0,00	6,51 ± 0,05	6,54 ± 0,05
JXLSCORE	9,05 ± 0,16	9,30 ± 0,09	9,30 ± 0,04	9,29 ± 0,09
GRAPPA	12,69 ± 0,01	12,70 ± 0,00	12,64 ± 0,06	12,69 ± 0,03
JPASSWORD	10,21 ± 0,06	10,29 ± 0,04	10,28 ± 0,06	10,34 ± 0,04
JUNIT	11,06 ± 0,04	11,09 ± 0,00	11,06 ± 0,05	11,08 ± 0,01

Tabela 19. *P-value* e tamanho de efeito para instâncias de pequeno porte

Instâncias	ILS_CMS > BL		ILS_CMS < AG_GGA		ILS_CMS > AG_GNE		AG_GGA > AG_GNE	
	PV	ES	PV	ES	PV	ES	PV	ES
JSTL	<0,001	84%	0,039	64%	0,163	59%	0,325	43%
MTUNIS	<0,001	93%	-	50%	<0,001	78%	<0,001	78%
ISPELL	0,010	31%	<0,001	22%	0,171	60%	<0,001	87%
JNANOXML	<0,001	70%	0,006	62%	<0,001	92%	<0,001	88%
JODAMONE	<0,001	78%	-	50%	<0,001	72%	<0,001	72%
JXLSREADER	0,001	67%	-	50%	<0,001	90%	<0,001	90%
RCS	<0,001	18%	0,004	29%	0,032	34%	0,643	46%
SEEMP	<0,001	87%	-	50%	<0,001	75%	<0,001	75%
APACHE	<0,001	96%	0,161	47%	<0,001	77%	<0,001	78%
BISON	<0,001	94%	<0,001	22%	0,007	69%	<0,001	87%
UDTJAVA	<0,001	99%	0,246	44%	<0,001	97%	<0,001	91%
JVAOCR	<0,001	93%	<0,001	28%	<0,001	85%	<0,001	94%
SERVLET	0,073	63%	0,039	35%	0,651	53%	0,023	67%
PFCDA_BASE	0,233	59%	0,028	66%	<0,001	77%	0,015	68%
FORMS	<0,001	100%	0,011	60%	<0,001	87%	<0,001	82%
JSCATTERPL	<0,001	98%	0,161	47%	<0,001	90%	<0,001	95%
JFLUID	<0,001	100%	<0,001	78%	<0,001	100%	0,001	76%
JXLSCORE	<0,001	94%	0,367	57%	0,026	67%	0,483	55%
GRAPPA	<0,001	92%	0,030	34%	<0,001	95%	<0,001	85%
JPASSWORD	<0,001	91%	<0,001	21%	0,779	52%	<0,001	77%
JUNIT	<0,001	82%	<0,001	78%	<0,001	85%	0,041	65%

Considerando-se as instâncias de médio porte, o ILS_CMS obteve melhor média de *MQ* em relação às demais heurísticas em 9 das 12 instâncias (Tabela 20). A busca local obteve melhor média que o ILS_CMS em 2 das 12 instâncias, mas a diferença entre as médias foi significativa para apenas 1 instância. O tamanho de efeito médio na comparação entre estas duas heurísticas foi 85% a favor do ILS_CMS. O algoritmo genético AG_GGA obteve melhor média em relação às demais heurísticas em apenas 2 instâncias, produzindo soluções com *MQ* significativamente superior ao algoritmo AG_GNE em 11 instâncias de médio porte. Na comparação entre o ILS_CMS e o AG_GGA, o primeiro obteve tamanho de efeito médio de 63%, saindo vencedor em 9 de 12 instâncias.

Tabela 20. Médias e desvios padrão do *MQ* para instâncias de médio porte

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)
XMLDOM	10,87 ± 0,04	10,86 ± 0,03	10,84 ± 0,07	10,88 ± 0,05
TINYTIM	12,26 ± 0,04	12,51 ± 0,02	12,31 ± 0,07	12,45 ± 0,05
JKARYOSCOPE	18,87 ± 0,03	18,98 ± 0,02	18,90 ± 0,05	18,96 ± 0,04
GAE_PLUGIN	17,32 ± 0,02	17,28 ± 0,02	17,29 ± 0,04	17,29 ± 0,04
JAVACC	10,45 ± 0,05	10,60 ± 0,05	10,43 ± 0,07	10,62 ± 0,06
JAVAGEOM	13,84 ± 0,04	14,07 ± 0,02	13,67 ± 0,08	14,01 ± 0,04
INCL	13,60 ± 0,02	13,61 ± 0,00	13,53 ± 0,04	13,58 ± 0,03
JDENDOGRAM	26,02 ± 0,02	26,07 ± 0,01	25,58 ± 0,12	26,03 ± 0,03
XMLAPI	18,74 ± 0,08	18,99 ± 0,04	18,49 ± 0,11	18,97 ± 0,08
JMETAL	12,31 ± 0,08	12,41 ± 0,03	11,87 ± 0,11	12,39 ± 0,09
DOM4J	18,40 ± 0,16	18,83 ± 0,10	17,97 ± 0,19	18,77 ± 0,13
PDF_RENDERER	21,34 ± 0,18	21,85 ± 0,14	21,19 ± 0,19	21,82 ± 0,12

Tabela 21. *P-value* e tamanho de efeito para instâncias de médio porte

INSTÂNCIA	ILS_CMS > BL		ILS_CMS > AG_GGA		ILS_CMS > AG_GNE		AG_GGA > AG_GNE	
	PV	ES	PV	ES	PV	ES	PV	ES
XMLDOM	0,548	46%	0,083	38%	0,426	56%	0,021	67%
TINYTIM	<0,001	100%	<0,001	87%	<0,001	100%	<0,001	95%
JKARYOSCO	<0,001	100%	0,014	64%	<0,001	94%	<0,001	84%
GAE_PLUGIN	<0,001	16%	0,026	33%	0,040	35%	0,830	48%
JAVACC	<0,001	100%	0,145	39%	<0,001	99%	<0,001	97%
JAVAGEOM	<0,001	100%	<0,001	93%	<0,001	100%	<0,001	100%
INCL	<0,001	76%	<0,001	81%	<0,001	97%	<0,001	82%
JDENDOGRA	<0,001	96%	<0,001	87%	<0,001	100%	<0,001	100%
XMLAPI	<0,001	100%	0,636	54%	<0,001	100%	<0,001	100%
JMETAL	<0,001	91%	0,790	52%	<0,001	100%	<0,001	100%
DOM4J	<0,001	98%	0,053	65%	<0,001	100%	<0,001	100%
PDF_RENDE	<0,001	99%	0,277	58%	<0,001	100%	<0,001	100%

Nas instâncias de grande porte, o ILS_CMS obteve média de *MQ* significativamente superior que a busca local em todas as instâncias (Tabela 22). O tamanho de efeito médio do ILS_CMS em relação ao da busca local é 99% entre as instâncias de grande porte. Considerando todas as configurações e as 13 instâncias de grande porte, o algoritmo ILS_CMS obteve melhor média em 9 instâncias, enquanto o AG_GGA obteve melhor média em apenas 4 instâncias. No entanto, o AG_GGA supera o AG_GNE em todas as instâncias com tamanho de efeito igual a 100%. Nas instâncias de grande porte, o tamanho de efeito do ILS_CMS na comparação com o AG_GGA é levemente superior (67%).

Tabela 22. Médias e desvios padrão do *MQ* para instâncias de grande porte

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)	Média (<i>MQ</i>)
JUNG_GRAPH	31,00 ± 0,17	31,52 ± 0,14	30,16 ± 0,34	31,55 ± 0,11
JCONSOLE	26,19 ± 0,06	26,51 ± 0,01	24,99 ± 0,19	26,43 ± 0,06
JUNG_VISUALIZATION	20,44 ± 0,17	20,88 ± 0,21	20,16 ± 0,16	21,05 ± 0,07
PFCDA_SWING	28,72 ± 0,07	28,99 ± 0,03	26,25 ± 0,30	28,85 ± 0,08
JPASSWORD_FULL	27,52 ± 0,06	27,89 ± 0,08	24,74 ± 0,23	27,87 ± 0,08
JML	17,20 ± 0,10	17,40 ± 0,05	15,28 ± 0,16	17,35 ± 0,09
NOTEPAD_FULL	29,07 ± 0,07	29,48 ± 0,05	25,47 ± 0,28	29,49 ± 0,10
POORMANS	33,84 ± 0,14	34,13 ± 0,03	28,79 ± 0,28	34,06 ± 0,12
LOG4J	30,94 ± 0,24	31,43 ± 0,21	26,71 ± 0,30	31,49 ± 0,18
JTREEVIEW	47,04 ± 0,22	47,59 ± 0,15	39,28 ± 0,47	47,55 ± 0,12
JACE	26,40 ± 0,09	26,63 ± 0,02	23,16 ± 0,17	26,59 ± 0,08
JAVAWS	38,12 ± 0,09	38,27 ± 0,02	29,73 ± 0,36	38,19 ± 0,09
RES_COBOL	15,75 ± 0,16	15,97 ± 0,01	12,86 ± 0,10	15,89 ± 0,06

Tabela 23. *P-value* e tamanho de efeito para instâncias de grande porte

INSTÂNCIA	ILS_CMS > BL		ILS_CMS > AG_GGA		ILS_CMS > AG_GNE		AG_GGA > AG_GNE	
	PV	ES	PV	ES	PV	ES	PV	ES
JUNG_GRAPH	<0,001	98%	0,636	46%	<0,001	100%	<0,001	100%
JCONSOLE	<0,001	100%	<0,001	94%	<0,001	100%	<0,001	100%
JUNG_VISUALIZATION	<0,001	94%	0,001	24%	<0,001	100%	<0,001	100%
PFCDA_SWING	<0,001	100%	<0,001	94%	<0,001	100%	<0,001	100%
JPASSWORD_FULL	<0,001	100%	0,176	60%	<0,001	100%	<0,001	100%
JML	<0,001	96%	0,005	71%	<0,001	100%	<0,001	100%
NOTEPAD_FULL	<0,001	100%	0,941	51%	<0,001	100%	<0,001	100%
POORMANS	<0,001	100%	0,006	71%	<0,001	100%	<0,001	100%
LOG4J	<0,001	95%	0,371	43%	<0,001	100%	<0,001	100%
JTREEVIEW	<0,001	98%	0,198	60%	<0,001	100%	<0,001	100%
JACE	<0,001	100%	0,007	70%	<0,001	100%	<0,001	100%
JAVAWS	<0,001	100%	<0,001	89%	<0,001	100%	<0,001	100%
RES_COBOL	<0,001	98%	<0,001	97%	<0,001	100%	<0,001	100%

Em termos de qualidade da solução, o ILS_CMS se mostrou superior quando comparado às demais heurísticas. Tanto no grupo de instâncias de menor porte quanto nas instâncias com maior número de módulos e dependências, o ILS_CMS obteve os melhores valores de MQ na maior parte das instâncias, demonstrando escalabilidade a medida que o tamanho das instâncias aumenta.

O tamanho do efeito médio em todas as instâncias do ILS_CMS em relação à busca local foi de 87% a favor do ILS_CMS. A segunda heurística com melhores resultados foi o algoritmo genético AG_GGA, com tamanho de efeito médio para todas as instâncias de 57% a favor do ILS_CMS. Comparando as duas abordagens de algoritmo genético percebe-se que o AG_GGA obteve um tamanho de efeito médio de 86% a seu favor.

Por fim, considerando todos os resultados observados nesta seção, podemos concluir que existem evidências de que o ILS_CMS pode encontrar melhores soluções para a formulação mono-objetivo do problema CMS do que a busca local subida de encosta com múltiplos reinícios (hoje aceita como melhor heurística para a formulação mono-objetivo) e que os algoritmos genéticos baseados nas representações GNE e GGA.

4.6.2 Comparação dos tempos de execução

Nesta seção as abordagens estudadas são comparadas em relação à eficiência. Foram obtidos os tempos de execução médios, em segundos, para cada uma das 40 instâncias. Os resultados foram divididos de acordo com o tamanho das instâncias (até 100 módulos, entre 101 e 200 módulos e maior que 200 módulos) nas Tabelas 24, 25 e 26, respectivamente.

Tabela 24. Tempo de execução médio para instâncias de pequeno porte.

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)
JSTL	≈ 0	0,01	0,08	0,09
MTUNIS	0,01	0,01	0,11	0,1
ISPELL	0,01	0,01	0,13	0,18
JNANOXML	0,01	0,01	0,13	0,24
JODAMONEY	0,01	0,01	0,16	0,28
JXLSREADER	0,01	0,01	0,17	0,30
RCS	0,02	0,02	0,23	0,34
SEEMP	0,01	0,02	0,24	0,47
APACHE	0,02	0,02	0,37	0,74
BISON	0,03	0,03	0,44	0,7
UDTJAVA	0,05	0,06	1,51	3,40
JAVAOOCR	0,04	0,05	1,60	3,94
SERVLET	0,04	0,05	1,89	4,87
PFCDA_BASE	0,06	0,08	2,38	6,15
FORMS	0,07	0,11	2,58	6,61
JSCATTERPLOT	0,07	0,10	3,19	9,27
JFLUID	0,10	0,15	4,49	13,64
JXLSCORE	0,11	0,14	4,76	14,22
GRAPPA	0,11	0,13	4,96	13,11
JPASSWORD	0,13	0,17	7,29	23,35
JUNIT	0,12	0,17	7,69	21,00

Tabela 25. Tempo de execução médio para instâncias de médio porte.

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)
XMLDOM	0,14	0,22	12,37	39,77
TINYTIM	0,28	0,38	21,47	62,70
JKARYOSCOPE	0,25	0,35	21,54	64,32
GAE_PLUGIN	0,25	0,35	22,23	73,55
JAVACC	0,39	0,55	34,12	107,46
JAVAGEOM	0,72	0,96	59,28	169,34
INCL	0,33	0,54	39,78	176,35
JDENDOGRAM	0,44	0,57	50,08	172,35
XMLAPI	0,37	0,55	52,24	201,79
JMETAL	0,69	0,90	73,19	235,73
DOM4J	0,63	0,83	75,56	253,51
PDF_RENDERER	0,54	0,76	73,33	277,31

Tabela 26. Tempo de execução médio para instâncias de grande porte.

	BL	ILS_CMS	AG_GNE	AG_GGA
INSTÂNCIA	Tempo (s)	Tempo (s)	Tempo (s)	Tempo (s)
JUNG_GRAPH	0,53	0,70	82,64	302,29
JCONSOLE	0,70	0,95	107,7	390,04
JUNG_VISUALIZATION	0,75	0,95	107,88	406,19
PFCDA_SWING	0,90	1,15	163,96	655,02
JPASSWORD_FULL	1,22	1,62	214,38	853,98
JML	1,49	1,91	232,37	902,34
NOTEPAD_FULL	1,45	1,88	297,09	1.268,55
POORMANS	1,27	1,75	296,34	1.342,54
LOG4J	1,29	1,84	310,29	1.447,39
JTREEVIEW	1,38	1,96	373,38	1.821,84
JACE	1,85	2,88	444,66	2.185,50
JAVAWS	1,95	2,96	606,47	3.164,09
RES_COBOL	8,37	12,36	1.932,39	9.336,93

Quanto ao critério de eficiência, pode-se observar nos resultados das tabelas acima que os algoritmos genéticos consumiram um tempo computacional muito superior aos tempos das demais heurísticas. Nas instâncias de grande porte os algoritmos genéticos consumiram, em média, cerca de mil vezes mais que a busca local. A diferença entre os tempos de execução dos diferentes algoritmos pode ser visualizada na Figura 4, que apresenta um gráfico com o tempo computacional gasto por cada heurística nas 46 instâncias. Neste gráfico, o eixo X representa o número de módulos, e o eixo Y representa o tempo de execução médio em segundos.

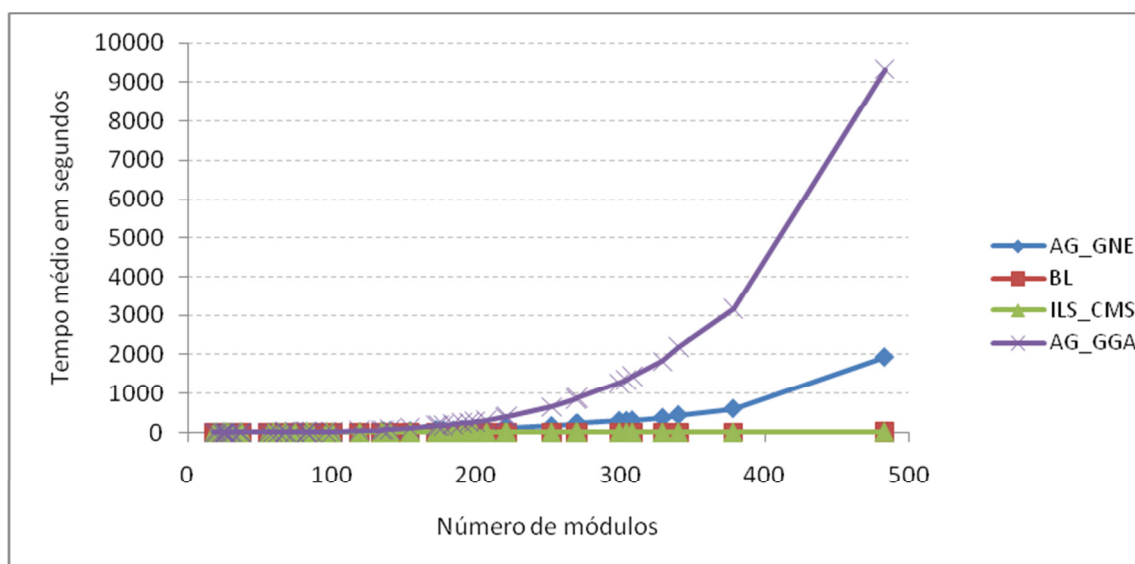


Figura 4. Tempo de execução dos algoritmos (em segundos) para diferentes tamanhos de instâncias (em número de módulos)

Em relação ao cálculo do valor da solução, na busca local não foi preciso realizar o cálculo completo do MQ a cada iteração, uma vez que os valores de CF dos clusters inalterados permaneciam os mesmos. No caso da busca local, a cada iteração somente um módulo era movimentado de um cluster para outro, o que afetava somente os valores de CF dos clusters origem e destino da movimentação. A rotina de cálculo percorre as dependências do módulo, realizando os ajustes necessários nos CF dos clusters origem e destino da movimentação. Entretanto, nos algoritmos genéticos, devido aos procedimentos de recombinação e mutação, com a modificação de diversos clusters, não foi possível adotar o cálculo de MQ Incremental (Incremental Turbo MQ), medida que ajudou a reduzir o tempo computacional nas heurísticas busca local e ILS_CMS. Entretanto, formas de tornar mais eficiente o cálculo do *fitness* nos algoritmos genéticos podem vir a serem viabilizadas em trabalhos futuros.

A análise comparativa dos tempos médios entre a busca local e o algoritmo ILS_CMS indica que os tempos variam de modo similar à medida que as instâncias aumentam em tamanho. Isto pode ser visualizado na Figura 5, que mostra os tempos em segundos obtidos pela busca local e pelo ILS_CMS para as 46 instâncias. Embora o ILS_CMS tenha consumido mais tempo computacional (em média 38% a mais nas instâncias de grande porte), a diferença em valores absolutos dos tempos obtidos pela busca local e pelo ILS_CMS foi de poucos segundos. Isto torna a ILS_CMS uma alternativa promissora, uma vez que retorna soluções de melhor qualidade com um custo computacional dentro da mesma ordem de grandeza que a busca local.

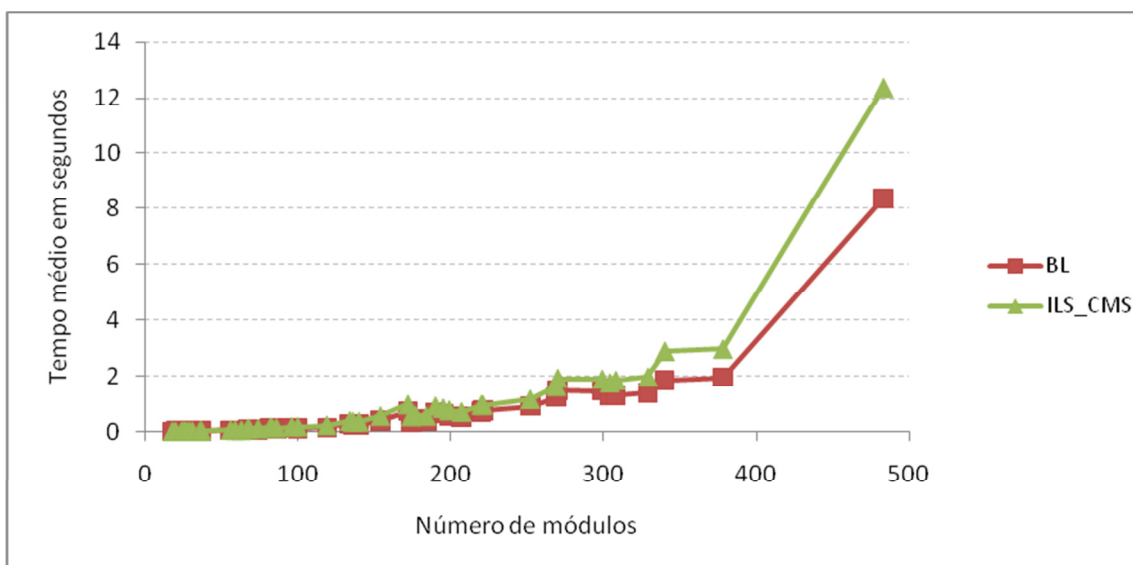


Figura 5. Evolução do tempo (seg.) da busca local e do ILS_CMS

Em relação ao tempo total, a execução das quatro heurísticas, 30 vezes para cada uma das 46 instâncias, gastou 283,89 horas (11,82 dias). A execução do primeiro experimento comparativo entre as 228 configurações do ILS consumiu um tempo total de 16,39 horas. As execuções das diferentes configurações do AG_GNE e AG_GGA do segundo experimento consumiram respectivamente 109,05 horas (4,54 dias) e 218,09 horas (9,08 dias).

4.7 Estudo comparativo do ILS_CMS com instâncias de pequeno e médio porte da literatura

Com o objetivo de comparar o ILS_CMS com outros resultados reportados na literatura e reduzir ameaças de validade interna dos experimentos relatados nas seções anteriores, a heurística ILS_CMS e uma variante ILS_CMS* foram executadas em seis instâncias não ponderadas utilizadas em estudos prévios (PRADITWONG, 2011, PRADITWONG et al., 2011, MANCORIDIS, 2002). Foi utilizado o critério de parada de $2000 * n^2$ avaliações. Cabe a ressalva que a melhor configuração do ILS_CMS obtida no primeiro estudo experimental utilizou o critério de parada de $200 * n^2$ avaliações e que as configurações utilizando o parâmetro *histórico* não apresentaram melhores valores de *MQ* que as configurações equivalentes que não utilizaram a estratégia. Portanto, neste estudo experimental, além da heurística ILS_CMS foi avaliada a variante ILS_CMS*, que tem como diferencial o fato de usar o parâmetro *histórico* de cinco iterações sem melhoria, na qual reinicia a etapa de busca a partir de uma solução aleatória.

A Tabela 27 apresenta as heurísticas da literatura e as variantes do ILS_CMS que foram comparadas. A primeira coluna exibe um identificador para a heurística, a segunda coluna descreve suas informações detalhadas e a última coluna apresenta a referência na literatura de onde os resultados a ser comparados foram colhidos.

Tabela 27. Heurísticas da literatura e variantes do ILS_CMS.

Heurística	Características	Referência
HC_LIT	<i>Random Restart Hill Climbing.</i>	(PRADITWONG et al. 2011)
GENE_LIT	Algoritmo genético mono-objetivo. Seleção: <i>binary-tournament.</i> Crossover: <i>one-point-crossover.</i> Mutação: <i>one-point mutation.</i> População: $10*n$.	(PRADITWONG, 2011)
GGA_LIT	Algoritmo genético mono-objetivo. Seleção: <i>binary tournament.</i> Crossover: GGA. Mutação: sem mutação. População: $10*n$.	(PRADITWONG, 2011)
ECA_LIT	Algoritmo genético multiobjetivo. Formulação ECA.	(PRADITWONG et al. 2011)
ILS_CMS	Solução inicial gulosa. Sem histórico. Perturbação de 10% baseada na movimentação de módulos.	-
ILS_CMS*	Solução inicial gulosa, Histórico = 5 iterações. Perturbação de 10% baseada na movimentação de módulos.	-

A Tabela 28 apresenta as médias e desvios padrão de *MQ* coletadas da literatura, além das médias obtidas pelo ILS_CMS e ILS_CMS*. Valores em negrito representam a melhor média por instância. O ILS_CMS* atingiu seis melhores médias enquanto o ILS_CMS atingiu três melhores médias. Entre as heurísticas da literatura somente o algoritmo genético multiobjetivo (ECA_LIT) atingiu uma das melhores médias.

Tabela 28. Resultados para 2000*n² avaliações.

	HC_LIT	GGA_LIT	GENE_LIT	ECA_LIT	ILS_CMS	ILS_CMS*
INSTÂNCIA	Média (MQ)	Média (MQ)	Média (MQ)	Média (MQ)	Média (MQ)	Média (MQ)
MTUNIS	2,25±0,060	2,23±0,051	2,29±0,026	2,31±0,000	2,31 ± 0,000	2,31 ± 0,000
ISPELL	2,34±0,022	2,34±0,016	2,35±0,014	2,34±0,022	2,36 ± 0,004	2,36 ± 0,000
RCS	2,22±0,020	2,23±0,029	2,26±0,020	2,24±0,022	2,27 ± 0,000	2,28 ± 0,000
BISON	2,64±0,041	2,23±0,051	2,29±0,026	2,64±0,029	2,69 ± 0,005	2,70 ± 0,007
GRAPPA	12,68±0,017	12,45±0,144	10,83±0,097	12,58±0,053	12,70 ± 0,000	12,70 ± 0,004
INCL	13,57±0,035	13,14±0,170	8,14±0,089	13,45±0,088	13,62 ± 0,008	13,64 ± 0,004

Não foram realizados testes estatísticos entre as heurísticas da literatura e o ILS_CMS uma vez não foram disponibilizados nas referências da literatura os resultados para cada ciclo de execução. A Tabela 29 apresenta a melhoria percentual das médias do ILS_CMS* em relação às heurísticas da literatura. A média calculada sobre o percentual de melhoria em relação aos resultados da literatura mostrou maiores

resultados em relação à versão de algoritmo genético mono-objetivo GNE_LIT (17,49%) seguida do algoritmo genético mono-objetivo GGA_LIT (5,60%). Também houve melhoria média em relação aos resultados reportados para o HC_LIT (1,53%), que utiliza o algoritmo de busca local subida de encosta com reinícios aleatórios, e para o algoritmo genético multiobjectivo ECA_LIT (1,14%). A diferença em termos percentuais entre o ILS_CMS e variante ILS_CMS* foi relativamente pequena para as instâncias consideradas (0,16%).

Tabela 29. Melhoria percentual do valor do MQ .

	ILS_CMS* e HC_LIT	ILS_CMS* e GGA_LIT	ILS_CMS* e GNE_LIT	ILS_CMS* e ECA_LIT	ILS_CMS* e ILS_CMS
	Melhoria (MQ)	Melhoria (MQ)	Melhoria (MQ)	Melhoria (MQ)	Melhoria (MQ)
MTUNIS	2,67%	3,59%	0,87%	0,00%	0,00%
ISPELL	0,85%	0,85%	0,43%	0,85%	0,00%
RCS	2,70%	2,24%	0,88%	1,79%	0,44%
BISON	2,27%	21,08%	17,90%	1,89%	0,37%
GRAPPA	0,16%	2,01%	17,27%	0,95%	0,00%
INCL	0,52%	3,81%	67,57%	1,34%	0,15%
MÉDIA	1,53%	5,60%	17,49%	1,14%	0,16%

A Tabela 30 apresenta o p -value e tamanho de efeito entre o ILS_CMS e a variante ILS_CMS*. Considerando todas as instâncias, a variante ILS_CMS* obteve um tamanho de efeito médio favorável de 70% em relação ao ILS_CMS. Este resultado pode indicar que para a quantidade dez vezes maior de avaliações o uso do *histórico* contribuiu para que o ILS_CMS não ficasse preso em ótimos locais.

Tabela 30. P -value e tamanho de efeito entre ILS_CMS* e ILS_CMS.

INSTÂNCIA	ILS_CMS* > ILS_CMS	
	PV	ES
MTUNIS	-	50%
ISPELL	0,005	62%
RCS	<0,001	100%
BISON	<0,001	87%
GRAPPA	0,233	42%
INCL	<0,001	84%

4.8 Características das soluções obtidas para as instâncias de testes

Os melhores valores de MQ obtidos pelo ILS_CMS estão listados na Tabela 31 a seguir, servindo de referência para trabalhos futuros. Nesta tabela são apresentadas as

instâncias, o valor de MQ , a quantidade total de interdependências entre os módulos de diferentes *clusters* (INTER), a quantidade total de dependências entre módulos contidos nos *clusters* (INTRA), a quantidade de clusters original (obtida dos códigos fontes), o total de clusters obtido pela melhor solução, o número de módulos do maior cluster (MÁX MÓDULO), o número de módulos do menor cluster (MÍN MÓDULO) e a quantidade de clusters com um único módulo (ÚNICO MÓDULO).

Tabela 31. Características das melhores soluções obtidas pelo ILS_CMS.

INSTÂNCIA	MQ	INTER	INTRA	TOTAL DE CLUSTERS ORIGINAL	TOTAL DE CLUSTERS OBTIDO	MÁX MÓDULO	MÍN MÓDULO	ÚNICO MÓDULO
JSTL	3,40	3	17	4	4	6	3	0
MTUNIS	2,31	30	27	-	5	5	3	0
ISPELL	2,36	77	26	-	8	4	2	0
JNANOXML	3,82	39	25	1	9	4	2	0
JODAMONEY	2,75	65	37	2	9	5	2	0
JXLSREADER	3,60	48	25	1	10	4	2	0
RCS	2,28	132	31	-	11	4	2	0
SEEMP	4,65	21	40	9	8	8	2	0
APACHE	5,77	47	39	2	13	5	2	0
BISON	2,70	136	43	-	12	5	2	0
UDTJAVA	5,28	172	55	7	20	6	2	0
JAVAOCR	9,02	84	71	13	20	7	2	0
SERVLET	9,56	78	53	4	22	5	2	0
PFCDA_BASE	7,34	147	50	8	24	5	1	1
FORMS	8,33	179	91	5	23	6	2	0
JSCATTERPLOT	10,74	132	100	1	25	6	2	0
JFLUID	6,58	224	91	4	24	13	2	0
JXLSCORE	9,38	243	87	10	27	6	2	0
GRAPPA	12,71	214	81	-	34	5	2	0
JPASSWORD	10,35	257	104	7	31	5	2	0
JUNIT	11,09	161	115	6	29	15	2	0
XMLDOM	10,92	106	103	9	26	38	2	0
TINYTIM	12,54	387	177	9	42	8	2	0
JKARYOSCOPE	18,99	276	184	1	43	6	2	0
GAE_PLUGIN	17,34	200	175	22	39	21	2	0
JAVACC	10,71	525	197	6	44	25	2	0
JAVAGEOM	14,10	1212	233	21	57	7	1	1
INCL	13,64	221	139	-	38	24	1	2
JDENDOGRAM	26,08	366	217	1	60	10	2	0
XMLAPI	19,10	257	156	17	51	38	2	0
JMETAL	12,47	980	157	46	68	5	2	0

DOM4J	18,95	740	190	16	70	5	2	0
PDF_RENDERER	21,97	323	306	10	53	41	2	0
JUNG_GRAPH_MODEL	31,64	397	206	21	80	5	2	0
JCONSOLE	26,52	584	275	4	75	8	2	0
JUNG_VISUALIZATION	21,16	701	218	11	78	7	1	1
PFCDA_SWING	29,08	609	276	37	91	9	2	0
JPASSWORD_FULL	28,02	1046	302	10	94	6	2	0
JML	17,50	1482	263	15	94	11	1	2
NOTEPAD_FULL	29,62	1026	323	50	106	8	1	2
POORMANS CMS	34,19	817	301	15	106	10	2	0
LOG4J	31,72	789	289	20	99	17	1	1
JTREEVIEW	47,82	668	389	11	110	6	1	3
JACE	26,71	977	547	12	73	51	1	1
JAWAWS	38,32	951	452	10	110	19	2	0
RES_COBOL	15,99	6546	617	12	122	31	1	7

Em relação às dependências entre módulos listadas na Tabela 31, em média as interdependências entre módulos de clusters distintos (INTER) corresponderam a 66% do total de dependências, enquanto as dependências entre módulos do mesmo cluster (INTRA) corresponderam a 34% do total de dependências. Outro fato que pode ser observado é que o número de clusters obtido pelas soluções (TOTAL DE CLUSTER OBTIDO) foi maior que o número de clusters obtido diretamente no código (TOTAL DE CLUSTER ORIGINAL). Considerando o número máximo de clusters que pode ser obtido como sendo igual ao número de módulos em cada instância (ou seja, um módulo por cluster), em média os clusters da solução original contém 8% do número máximo possível enquanto nas soluções obtidas a média foi de 32%.

Este incremento no número de clusters pode indicar um aumento de complexidade no modelo do software. Estudos recentes (BARROS e FARZAT, 2013, BARROS, 2013) indicaram que algumas métricas tendem a acarretar aumento significativo do número de clusters. Entretanto um grande número de pacotes com poucas classes pode não ser um modelo de software recomendável (LANZA e MARINESCU, 2006). Conforme tratado na Seção 2.6, uma abordagem possível para limitar o aumento do número de clusters, é a melhoria através de pequenas mudanças incrementais no software, restringindo as alterações a serem realizadas. Outra abordagem que pode ser usada é integrar o conhecimento dos desenvolvedores, de forma que o modelo seja aprimorado de forma semiautomática, com a interação humana (Seção 2.5).

Em relação ao número de clusters com um único módulo, este número foi relativamente pequeno. Um número alto poderia indicar uma reorganização do software de baixa qualidade. Das 46 soluções, 36 soluções não continham clusters com um único módulo. As soluções que possuíam cluster com um único módulo, obtiveram uma quantidade pequena de clusters com um único módulo. O maior número foi na solução obtida na instância com maior número de módulos (RES_COBOL) com sete clusters.

4.9 Ameaças à validade do estudo

Uma questão fundamental concernente aos experimentos diz respeito a quão válidos são os seus resultados. Segundo WOHLIN et al. (2000), as ameaças à validade podem ser divididas em quatro categorias: Construção, Interna, Externa e Conclusão.

A validade de construção trata da relação entre a teoria e observação. Como medida contra a ameaça da validade de construção, a definição teórica do problema foi tratada no Capítulo 1 e o projeto experimental no Capítulo 4. Também foram discutidas as métricas de eficiência (tempo de execução) e eficácia (valor de MQ) a serem utilizadas para responder às questões de pesquisa propostas na Seção 4.1.

Caso seja observado um relacionamento entre o tratamento e a saída, a validação interna deve assegurar que se trata de uma relação causal, e não o resultado de um fator que não se tem controle ou não pode ser medido. Embora não tenham sido utilizados pacotes tais como o JMetal (DURILLO e NEBRO, 2011), que disponibiliza algumas heurísticas prontas, os valores de MQ obtidos são comparáveis com estudos prévios (BARROS, 2012, PRADITWONG, 2011, PRADITWONG et al., 2011). A Seção 4.7 realizou uma análise comparativa dos resultados obtidos na literatura com os resultados obtidos pela heurística ILS_CMS. Como medida contra as ameaças de validade interna, também foram disponibilizados os códigos-fonte e instâncias utilizadas nas heurísticas no link <https://github.com/cmsp/ils/>. As soluções com valores máximos de MQ obtidos para cada instância foram descritas na Seção 4.8.

As ameaças de validade externa relacionam-se com a generalização e se o estudo pode ser usado fora do escopo definido. Embora as instâncias tenham sido coletadas principalmente de softwares escritos na linguagem Java, os casos consistiram de aplicações reais com objetivos diversos e tamanhos variados, agrupadas em instâncias de pequeno, médio e grande porte. Também foram descritos os algoritmos e parâmetros utilizados (Capítulo 3), de modo a permitir a replicação do estudo. As instâncias

utilizadas foram descritas na Seção 4.1, bem como a ferramenta e procedimentos efetuados para a coleta.

A validade de conclusão está relacionada com a relação entre o tratamento e os resultados e em certificar-se deste relacionamento com certo nível de significância estatística. Entre as medidas para combater às ameaças de conclusão, foi considerada a natureza aleatória das heurísticas de busca local, ILS_CMS e algoritmos genéticos, executando as heurísticas 30 vezes para cada instância. Outra medida contra as ameaças de conclusão foi adotar testes de inferência estatísticos. Para verificar o nível de significância em que as amostras são distintas entre si foi adotado o teste não paramétrico Wilcoxon-Mann-Whitney. Para verificar quanto uma amostra tende a ser superior a outra, foi calculado o tamanho de efeito (Vargha-Delaney). Outra medida contra ameaças à validade da conclusão foi adotar como base de comparação a busca local usada em outros estudos prévios dentro do tema CMS, conforme levantamento efetuado na revisão bibliográfica (Capítulo 2).

4.10 Considerações finais

Este capítulo apresentou os resultados dos experimentos computacionais realizados para avaliar a heurística ILS_CMS, juntamente com as avaliações realizadas em cada experimento. O próximo capítulo descreverá as contribuições, limitações do presente trabalho e perspectivas de melhoria futura.

Capítulo 5 - Conclusões

5.1 Contribuições

Os resultados dos experimentos apresentados no Capítulo 4 indicam que o ILS_CMS demonstrou ser uma heurística eficiente e eficaz para a resolução do problema CMS, comparativamente a outras heurísticas frequentemente usadas na comunidade de Engenharia de Software baseada em Buscas. Em termos de eficácia o ILS_CMS superou o algoritmo de busca local subida de encosta, com um tamanho de efeito médio de 87% a seu favor. Em relação ao algoritmo genético AG_GGA, ambos se mostraram eficazes para resolução do problema CMS. Considerando todas as instâncias o ILS_CMS teve a seu favor um tamanho de efeito médio levemente superior de 57%. Entretanto, em termos de eficiência (tempo de execução) superou em grande margem os algoritmos genéticos. Ainda em relação à eficiência se mostrou competitivo em relação ao algoritmo de busca local, com tempo de execução (medido em segundos) dentro da mesma ordem de magnitude, considerando instâncias até 483 módulos.

Além da avaliação da heurística ILS_CMS, cada um dos experimentos em específico apresentou contribuições (avaliações efetuadas sobre as instâncias do problema CMS) que servem de referência para trabalhos futuros. No primeiro experimento foram realizados estudos com diferentes configurações do algoritmo proposto ILS_CMS. Os resultados indicaram que as configurações do ILS_CMS que utilizaram o algoritmo guloso para geração da solução inicial tiveram melhor eficácia que as configurações utilizando uma solução inicial aleatória. O uso das estratégias de movimentação e troca de módulos se mostrou mais eficiente que as estratégias de união e divisão de clusters, com a ressalva que o percentual foi mantido fixo durante a execução, sendo o mesmo para as diferentes estratégias.

O segundo experimento envolveu algoritmos genéticos com diferentes operadores de recombinação. Como contribuição os resultados indicaram que as configurações com o operador de recombinação *two-point-crossover* obtiveram maior eficácia que o operador de recombinação *one-point-crossover*, mantendo-se as demais

características equivalentes. O *one-point-crossover* tem sido mais frequentemente utilizado nos trabalhos envolvendo algoritmos genéticos, conforme levantamento bibliográfico (Capítulo 2). Ainda em relação aos estudos efetuados entre os operadores genéticos, os resultados indicam que adicionar estratégias baseadas em agrupamentos (FALKENAUER, 1996), com a utilização de operadores de recombinação e mutação atuando sobre os clusters em vez dos módulos, aumenta a eficácia dos resultados em relação ao uso exclusivo de operadores genéticos, tais como a movimentação dos módulos. Os resultados mostraram que a melhor configuração do algoritmo genético AG_GGA obteve resultados superiores aos obtidos pela busca local. Foram introduzidas duas modificações sobre a proposta de PRADITWONG (2011): operadores de mutação atuando sobre os clusters e uma restrição sobre a quantidade de clusters atingidos pelos operadores genéticos. Os resultados indicaram que as propostas retornaram soluções mais eficazes que as versões sem operador de mutação e sem a restrição.

No terceiro experimento foram comparadas a melhor configuração do ILS_CMS, as melhores configurações dos algoritmos genéticos e a busca local com múltiplos reinícios aleatórios. Em termos de qualidade da solução, a ILS_CMS obteve melhores médias nos valores de MQ e melhor tamanho de efeito médio em relação às demais heurísticas. Um dos passos do algoritmo ILS_CMS utiliza o mesmo procedimento de busca local utilizado na versão com múltiplos reinícios, com as seguintes diferenças: i) o ILS_CMS parte de uma solução inicial obtida por um algoritmo guloso, e ii) após a busca local, em vez do reinício em uma solução gerada de forma aleatória, é aplicada uma perturbação baseada na movimentação de um percentual dos módulos selecionados aleatoriamente. Portanto, uma vez implementada a heurística de busca local, é relativamente fácil implementar o procedimento de geração da solução inicial e o método de perturbação, incrementando a qualidade das soluções.

O último experimento avaliou os resultados obtidos pelo ILS_CMS com resultados obtidos por outros autores e publicados na literatura técnica da área de Engenharia de Software baseada em Buscas, considerando seis instâncias representadas como grafos não ponderados. A avaliação também indicou que os resultados obtidos pelo algoritmo ILS_CMS foram superiores aos resultados da busca local com múltiplos reinícios e dos algoritmos genéticos, em linha com os resultados obtidos no terceiro experimento.

5.2 Limitações e perspectivas futuras do trabalho

Uma sugestão para aprimoramentos futuros do algoritmo proposto ILS_CMS é ajustar o método de perturbação durante a execução, de acordo com as etapas de intensificação e diversificação, e conforme as diferentes estratégias. Uma vez que as estratégias de união, divisão e explosão atuam sobre os clusters e as estratégias de movimentação e troca atuam sobre os módulos, as estratégias selecionadas para a perturbação e os percentuais de módulos e clusters a ser afetados podem ser alterados de acordo com a intensidade do método de perturbação almejado em uma etapa da execução. Outra análise que pode ser realizada é a verificação das configurações mais promissoras para cada classe de instâncias, isto é, se ocorre variação da configuração vencedora para instâncias de diferentes tamanhos e características. Para o aperfeiçoamento do ILS_CMS também é possível avaliar a variação do critério de aceitação durante a execução, de forma similar ao critério de aceitação do *Simulated Annealing*.

Também é indicado avaliar possibilidades para o aperfeiçoamento da estratégia de histórico, através do melhor ajuste deste parâmetro durante a execução do algoritmo ILS_CMS. Pelos resultados do quarto estudo experimental apresentado no Capítulo 4, percebemos que algumas instâncias podem ser beneficiar do uso de histórico quando o número de avaliações disponível para o algoritmo foi suficientemente grande.

A análise das soluções obtidas pelo ILS_CMS efetuada na Seção 4.8 indicou um aumento significativo do número de clusters obtidos nas soluções em relação ao número de clusters existentes no software original. Como perspectivas futuras o ILS_CMS poderia incorporar uma abordagem incremental e interativa, além de utilizar métricas distintas da MQ para avaliação da clusterização. Por exemplo, alguns estudos tem utilizado acoplamento lógico⁵ e léxico⁶ para avaliar a estrutura do software. OLIVA et al. (2013) avaliou a degradação do modelo do software ao longo de sucessivas manutenções, analisando o acoplamento lógico dos componentes a partir do histórico de *commits* obtido no repositório de controle de versões do código-fonte. Em outra

⁵ O acoplamento lógico entre os componentes pode ser obtido a partir de uma ferramenta de controle de versionamento do código-fonte, com os componentes usualmente alterados em conjunto tendo maior acoplamento.

⁶ O acoplamento léxico pode ser obtido, por exemplo, através da terminologia nos comentários e o nome utilizado nas classes.

abordagem, VASCONCELOS e WERNER (2007) fazem uma análise léxica, recuperando a arquitetura dos componentes a partir de conceitos relacionados dentro do domínio da aplicação.

Por fim, a versão básica de um algoritmo baseado na meta-heurística ILS é simples de implementar e, como mostrado, é capaz de produzir resultados de melhor qualidade que a busca local com reinícios aleatórios, consumindo esforço computacional comparável. Acredita-se também, como perspectivas de trabalhos futuros, que a meta-heurística ILS possa ser uma boa escolha para tratar outros problemas da área de Engenharia de Software baseada em Buscas (*Search Based Software Engineering - SBSE*).

Referências

- ALBA, E.; DORRONSORO, B. **Cellular Genetic Algorithms**. 1st. ed. Berlim: Springer Publishing Company, Incorporated, 2008.
- AMOUI, M.; MIRARAB, S.; ANSARI, S.; LUCAS, C. “A Genetic Algorithm Approach to Design Evolution Using Design Pattern Transformation”. **International Journal of Information Technology and Intelligent Computing**, v. 1, n. 2, pp. 235-244, 2006.
- ANQUETIL, N.; FOURRIER, C.; LETHBRIDGE, T. C. “Experiments with Clustering As a Software Remodularization Method”. In: **Proceedings of the Sixth Working Conference on Reverse Engineering**, pp. 235-255. Washington, DC, USA, Oct. 1999.
- BARROS, M. D. O. “An analysis of the effects of composite objectives in multiobjective software module clustering”. In: **Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12**, p. 1205, Philadelphia, Pennsylvania, USA, 2012.
- BARROS, M. D. O. “An Experimental Study on Incremental Search-Based Software Engineering”. In: **Proceedings of the Search Based Software Engineering - 5th International Symposium, SSBSE 2013**, v. 8084, pp. 34–49, St. Petersburg, Russia, August 24-26, 2013. Springer.
- BARROS, M. D. O.; FARZAT, F. D. A. “What Can a Big Program Teach Us about Optimization?” In: **Search Based Software Engineering SE - 24**. v. 8084, pp. 275–281, 2013. Springer Berlin Heidelberg.
- BAVOTA, G.; CARNEVALE, F.; DE LUCIA, A.; DI PENTA, M.; OLIVETO, R. “Putting the developer in-the-loop: an interactive GA for software re-modularization”. In: **Proceedings of the 4th international conference on Search Based Software Engineering**, pp. 75–89, 2012. Springer-Verlag.
- BLUM, C.; ROLI, A. “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison”. **ACM Comput. Surv.**, v. 35, n. 3, pp. 268–308, 2003. New York, NY, USA: ACM.
- BOTELHO, A.L.V., SEMAAN, G.S., OCHI, L.S. “Agrupamento de Sistemas Orientados a Objetos com Metaheurísticas Evolutivas”. In: **O Simpósio Brasileiro de Sistemas de Informação (SBSI)**, Salvador, Brasil, 2011.
- BRIAND, L. C.; MORASCA, S.; BASILI, V. R. “Defining and validating measures for object-based high-level design”. **IEEE Transactions on Software Engineering**, v. 25, n. 5, pp. 722-743. 1999.
- CHIDAMBER, S. R.; KEMERER, C. F. “A metrics suite for object oriented design”. **Software Engineering, IEEE Transactions on**, v. 20, n. 6, pp. 476–493, 1994.
- CLAUDE, B. **Perfect graphs**. Six Papers on Graph Theory. Calcutta: Indian Statistical Institute, pp. 1–21, 1963.
- CLAUSET, A.; NEWMAN, M. E. J.; MOORE, C. “Finding community structure in very large networks”. **Physical review E**, v. 70, n. 6, pp. 1-6, 2004.
- COLLINS, R. J.; JEFFERSON, D. R. “Selection in massively parallel genetic algorithms”. In: **Proceedings of the 4th International Conference on Genetic Algorithms and their application (ICGA)**. pp. 249–256, San Diego, CA., 1991.

- CORMACK, R. M. “A Review of Classification”. **Journal of the Royal Statistical Society. Series A (General)**, v. 134, n. 3, pp. 321–367, 1971. Wiley for the Royal Statistical Society.
- COWLING, P. I.; KENDALL, G.; SOUBEIGA, E. “A Hyperheuristic Approach to Scheduling a Sales Summit”. **Lecture Notes in Computer Science. Practice and Theory of Automated Timetabling III**, v. 2079, pp. 176–190, 2001. Springer.
- DEB, K.; KALYANMOY, D. **Multi-Objective Optimization Using Evolutionary Algorithms**. New York, NY, USA: John Wiley & Sons, Inc., 2001.
- DIAS, C. R.; OCHI, L. S. “Efficient Evolutionary Algorithms for the Clustering Problem in Directed Graphs”. In: **Proceedings of the 2003 IEEE Congress on Evolutionary Computation**. v.1, pp. 983–988, 2003.
- DIAS, C. **Algoritmos evolutivos para o Problema de Clusterização de Grafos Orientados: Desenvolvimento e Análise Experimental**, Dissertação, Universidade Federal Fluminense, 2004.
- DOVAL, D.; MANCORIDIS, S.; MITCHELL, B. S. “Automatic clustering of software systems using a genetic algorithm”. **STEP '99. Proceedings Ninth International Workshop Software Technology and Engineering Practice**. pp. 73–81, 1999.
- DURILLO, J. J.; NEBRO, A. J. “jMetal: A Java Framework for Multi-objective Optimization”. **Adv. Eng. Softw.**, v. 42, n. 10, pp. 760–771, 2011. Oxford, UK, UK: Elsevier Science Ltd.
- FALKENAUER, E. “A Hybrid Grouping Genetic Algorithm for Bin Packing”, **Journal of Heuristics**, v. 2, pp. 5–30, 1996.
- FELTOVICH, N. “Nonparametric Tests of Differences in Medians: Comparison of the WilcoxonMann-Whitney and Robust Rank-Order Tests”, mimeo. **Experimental Economics**, pp. 6–273, 2003.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
- GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman, 1979.
- GIBBS, S. J.; TSICHRITZIS, D.; CASAIS, E.; NIERSTRASZ, O.; PINTADO, X. “Class Management for Software Communities”. **Commun. ACM**, v. 33, n. 9, pp. 90–103, 1990.
- GLOVER, F.; LAGUNA, M. **Tabu Search**. Norwell, MA, USA: Kluwer Academic Publishers, 1997.
- GLOVER, F.; KOCHENBERGER, G. A. **Handbook of Metaheuristics**. Kluwer Academic Publishers, 2003.
- GORDON, A. D. “A Review of Hierarchical Classification”. **Journal of the Royal Statistical Society. Series A (General)**, v. 150, n. 2, pp. 119–137, 1987. Blackwell Publishing for the Royal Statistical Society.
- HALL, M.; MCMINN, P. “An Analysis of the Performance of the Bunch Modularisation Algorithm’s Hierarchy Generation Approach”. **4 th Symposium on Search Based-Software Engineering**, p. 19, 2012.
- HALL, M.; MCMINN, P.; WALKINSHAW, N. “Supervised Software Modularisation”. In: **Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)**. pp. 472–481, 2012. Washington, DC, USA: IEEE Computer Society.

- HARMAN, M.; JONES, B. F. “Search-Based Software Engineering”. **Information and Software Technology**, v. 43, pp. 833–839, 2001.
- HARMAN, M.; HIERONS, R., PROCTOR, M. “A new representation and crossover operator for search-based optimization of software modularization”. In: **GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference**. pp. 1351–1358, 2002. Morgan Kaufmann Publishers.
- HARMAN, M.; SWIFT, S.; MAHDAVI, K. “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**, p. 1029, New York, New York, USA: ACM Press., 2005.
- HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. “Search-based software engineering”. **ACM Computing Surveys**, v. 45, n. 1, pp. 1–61, New York, NY, USA: ACM., 2012.
- HENDERSON-SELLERS, B. **Object-Oriented Metrics. Measures of Complexity**. Upper Saddle River, NJ, USA: Prentice Hall, 1996.
- HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**. Cambridge, MA, USA: MIT Press, 1992.
- KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. “Optimization by simulated annealing”. **SCIENCE**, v. 220, n. 4598, p. 671–680, 1983.
- KLEINBERG, J.; TARDOS, E. **Algorithm Design**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., p. 679, 2005.
- KÖHLER, V.; FAMPA, M.; ARAÚJO, O. “Mixed-Integer Linear Programming Formulations for the Software Clustering Problem”. **Computational Optimization and Applications**, v. 55, n. 1, p. 113–135, 2013.
- KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA, USA: MIT Press, 1992.
- KUMARI, A. C. SRINIVAS, K. “Software Module Clustering using a Fast Multi-objective Hyper-heuristic Evolutionary Algorithm”. **International Journal of Applied Information Systems**, v. 5, n. 6, pp. 12–18, 2012.
- LANZA, M., MARINESCU, R., “**Object-oriented Metrics in Practice: using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems**”, Springer-Verlag, Berlin, Heidelberg, 2006
- LARMAN, C. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process**. 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- LEHMAN, M. M. “On understanding laws, evolution, and conservation in the large-program life cycle”. **Journal of Systems and Software**, v. 1, pp. 213–221, 1979. New York, NY, USA: Elsevier Science Inc.
- LEWIS, R.; PULLIN, E. “Revisiting the restricted growth function genetic algorithm for grouping problems”. **Evolutionary computation**, v. 19, n. 4, pp. 693–704, 2011.
- LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. “Iterated local search. Handbook of Metaheuristics”. **International Series in Operations Research and Management Science**, v. 57, pp. 321–353, 2002. Kluwer Academic Publishers.
- LUTZ, R. “Evolving good hierarchical decompositions of complex systems”. **Journal of Systems Architecture**, v. 47, n. 7, pp. 613–634, 2001. New York, NY, USA: Elsevier North-Holland, Inc.

- MAHDAVI, K.; HARMAN, M.; HIERONS, R. M. “A Multiple Hill Climbing Approach to Software Module Clustering”. **Proceedings of the International Conference on Software Maintenance**. pp. 315–324, 2003. Washington, DC, USA: IEEE Computer Society.
- MANCORIDIS, S.; MITCHELL, B. S.; RORRES, C.; CHEN, Y.; GANSNER, E. R. “Using Automatic Clustering to Produce High-Level System Organizations of Source Code”. In: **Proceedings of the 6th International Workshop on Program Comprehension**. pp. 45–53, 1998. Washington, DC, USA: IEEE Computer Society.
- MANCORIDIS, S.; MITCHELL, B. S.; CHEN, Y.; GANSNER, E. R. “Bunch: a clustering tool for the recovery and maintenance of software system structures”. In: **Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99)**. pp. 50–59, 1999. IEEE.
- MARTIN, R. C.. **Design Principles and Design Patterns**. 2000. Disponível em: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. Acesso em 03 maio 2014.
- MCCONNELL, S. **Code Complete, Second Edition**. Redmond, WA, USA: Microsoft Press, 2004.
- MITCHELL, B. S.; MANCORIDIS, S. “Comparing the decompositions produced by software clustering algorithms using similarity measurements”. In: **Proceedings IEEE International Conference on Software Maintenance. ICSM 2001**. pp. 744–753, 2001. IEEE Comput. Soc.
- MITCHELL, B. S. **A Heuristic Search Approach to Solving the Software Clustering Problem**, Ph D. thesis., Philadelphia, PA, USA: Drexel University, 2002.
- MITCHELL, B. S.; MANCORIDIS, S. “Using Heuristic Search Techniques To Extract Design Abstractions From Source Code”. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. pp. 1375–1382, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.
- MITCHELL, B. S.; MANCORIDIS, S. “On the automatic modularization of software systems using the Bunch tool”. **IEEE Transactions on Software Engineering**, v. 32, n. 3, pp. 193–208, 2006. Piscataway, NJ, USA: IEEE Press.
- MITCHELL, B. S.; MANCORIDIS, S. “On the evaluation of the Bunch search-based software modularization algorithm”. **Soft Computing**, v. 12, n. 1, pp. 77–93, 2007. Berlin, Heidelberg: Springer-Verlag.
- OBJECT MANAGEMENT GROUP, **Unified modeling language resource page**. 2012. Disponível em: <http://www.uml.org/>. Acesso em 03 maio 2014.
- OLIVA, G. A.; STEINMACHER, I.; WIESE, I.; GEROSA, M. A. “What Can Commit Metadata Tell Us About Design Degradation?” **Proceedings of the 2013 International Workshop on Principles of Software Evolution**. pp.18–27, 2013. New York, NY, USA: ACM.
- OSMAN, I.; LAPORTE, G. “Metaheuristics: A bibliography”. **Annals of Operations Research**, v. 63, n. 5, pp. 511–623, 1996. Springer Netherlands.
- PRADITWONG, K.; YAO, X. “A New Multi-objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm”. **2006 International Conference on Computational Intelligence and Security**. v. 1, pp. 286–291, 2006. IEEE.
- PRADITWONG, K. “Solving software module clustering problem by evolutionary algorithms”. **2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)**, pp. 154–159, 2011. IEEE.

- PRADITWONG, K.; HARMAN, M.; YAO, X. “Software Module Clustering as a Multi-Objective Search Problem”. **IEEE Transactions on Software Engineering**, v. 37, n. 2, pp. 264–282, 2011. IEEE.
- RADCLIFFE, N. J. “Formal analysis and random respectful recombination”. In R.K. Belew and L. B. Booker (Eds.), In: **Proceedings of the Fourth International Conference on Genetic Algorithms**, pp. 222-220, 1991. San Mateo, CA: Morgan Kaufmann.
- RÄIHÄ, O. “A survey on search-based software design”. **Computer Science Review**, v. 4, n. 4, pp. 203–249, 2010. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V.
- RISSANEN, J. “Modeling by shortest data description”. **Automatica**, v. 14, n. 5, pp. 465–471, 1978.
- SEMAAN, G. S.; OCHI, L. S. “Algoritmo Evolutivo para o Problema de Clusterização em Grafos Orientados”. In: **Simpósio de Pesquisa Operacional e Logística da Marinha**, Rio de Janeiro, Brasil. X SPOLM, 2007.
- SEMAAN, G.S., BOTELHO, S.L.V., OCHI, L.S. “Heurística Baseada em Busca Local Iterada para a resolução do Problema de Agrupamento de Sistemas Orientados a Objetos”. **XLIII SBPO (Simpósio Brasileiro de Pesquisa Operacional)**, Ubatuba - SP, 2011.
- SHTERN, M.; TZERPOS, V. “A Framework for the Comparison of Nested Software Decompositions”. **WCRE**. pp. 284–292, 2004. IEEE Computer Society.
- SÍMA, J.; SCHAEFFER, S. E. “On the NP-Completeness of Some Graph Cluster Measures”. **Proceedings of the 32Nd Conference on Current Trends in Theory and Practice of Computer Science**. p.530–537, 2006. Berlin, Heidelberg: Springer-Verlag.
- SIMONS, C. L.; PARMEE, I. C.; GWYNLLYW, R. “Interactive, Evolutionary Search in Upstream Object-Oriented Class Design”. **IEEE Transactions on Software Engineering**, v. 36, n. 6, pp. 798–816, 2010. Los Alamitos, CA, USA: IEEE Computer Society.
- SIMONS, C. L.; PARMEE, I. C. “Elegant Object-Oriented Software Design via Interactive, Evolutionary Computation”. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 42, n. 6, pp. 1797–1805, 2012.
- STÜTZLE, T. **Applying iterated local search to the permutation flow shop problem**. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt. 1998.
- STÜTZLE, T. **Local search algorithms for combinatorial problems: analysis, improvements, and new applications**. DISKI - Dissertationen zur Künstlichen Intelligenz. infix, Sankt Augustin, Germany. 1999.
- TUCKER, A.; SWIFT, S.; LIU, X. “Variable Grouping in Multivariate Time Series via Correlation”. **Trans. Sys. Man Cyber. Part B**, v. 31, n. 2, pp. 235–245, Piscataway, NJ, USA, 2001. IEEE Press.
- TUCKER, A.; CRAMPTON, J.; SWIFT, S. “RGFGA: An Efficient Representation and Crossover for Grouping Genetic Algorithms”. **Evolutionary Computation**, v. 13, n. 4, pp. 477–499, 2005.
- TZERPOS, V.; HOLT, R. C. “MoJo: A Distance Metric for Software Clusterings”. In: **Proceedings of the Sixth Working Conference on Reverse Engineering**. pp. 187–193, 1999. Washington, DC, USA: IEEE Computer Society.
- TZERPOS, V.; HOLT, R. C. “ACDC: An Algorithm for Comprehension-Driven Clustering”. In: **Proceedings of the Seventh Working Conference on Reverse**

- Engineering (WCRE'00)**, pp. 258–267, Washington, DC, USA, 2000. IEEE Computer Society.
- VARGHA, A.; DELANEY, H. D. “A Critique and Improvement of the “CL” Common Language Effect Size Statistics of McGraw and Wong”. **Journal of Educational and Behavioral Statistics**, v. 25, n. 2, pp. 101–132, 2000. American Educational Research Association and American Statistical Association.
- VASCONCELOS, A.; WERNER, C. “Architecture Recovery and Evaluation Aiming at Program Understanding and Reuse“. **Proceedings of the Quality of Software Architectures 3rd International Conference on Software Architectures, Components, and Applications**. Anais...Berlin, Heidelberg: Springer-Verlag, 2007.
- VOSS, S.; OSMAN, I. H.; ROUCAIROL, C. **Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization**. Norwell, MA, USA: Kluwer Academic Publishers, 1999.
- WEN, Z.; TZERPOS, V. “An Optimal Algorithm for MoJo Distance”. In: **Proceedings of the 11th IEEE International Workshop on Program Comprehension**. pp. 227–235, Washington, DC, USA , 2003. IEEE Computer Society.
- WEN, Z.; TZERPOS, V. “An effectiveness measure for software clustering algorithms”. In: **Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004**, 2004.
- WOHLIN, C.; RUNESON, P.; HÖST, M.; et al. **Experimentation in Software Engineering: An Introduction**. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- WOOD, J. A. **Improving Software Designs Via the Minimum Description Length Principle**. Ph.D. Thesis. University of Sussex, 1998.
- YOURDON, E.; CONSTANTINE, L. L. **Structured design: fundamentals of a discipline of computer program and systems design**. Upper Saddle River, NJ, USA: Yourdon Press, 1978.

Anexo I – Configurações do ILS_CMS

ID	Inicial	Mem.	Perturbação	Moves	Swaps	Splits	Joins	Exp.		ID	Inicial	Mem.	Perturbação	Moves	Swaps	Splits	Joins	Exp.
ILS1	Gulosa	-	Sequencial	10%	-	-	-	-		ILS58	Gulosa	Sim	Sequencial	-	-	10%	-	10%
ILS2	Gulosa	-	Sequencial	10%	10%	-	-	-		ILS59	Gulosa	Sim	Sequencial	-	-	-	10%	-
ILS3	Gulosa	-	Sequencial	10%	10%	10%	-	-		ILS60	Gulosa	Sim	Sequencial	-	-	-	10%	10%
ILS4	Gulosa	-	Sequencial	10%	10%	10%	10%	-		ILS61	Gulosa	Sim	Sequencial	-	-	-	-	10%
ILS5	Gulosa	-	Sequencial	10%	10%	10%	-	10%		ILS62	Gulosa	Sim	Sequencial	10%	10%	10%	10%	10%
ILS6	Gulosa	-	Sequencial	10%	10%	-	10%	-		ILS63	Gulosa	-	Aleatória	10%	10%	-	-	-
ILS7	Gulosa	-	Sequencial	10%	10%	-	10%	10%		ILS64	Gulosa	-	Aleatória	10%	10%	10%	10%	-
ILS8	Gulosa	-	Sequencial	10%	10%	-	-	10%		ILS65	Gulosa	-	Aleatória	10%	10%	10%	10%	-
ILS9	Gulosa	-	Sequencial	10%	-	10%	-	-		ILS66	Gulosa	-	Aleatória	10%	10%	10%	-	10%
ILS10	Gulosa	-	Sequencial	10%	-	10%	10%	-		ILS67	Gulosa	-	Aleatória	10%	10%	-	10%	-
ILS11	Gulosa	-	Sequencial	10%	-	10%	10%	10%		ILS68	Gulosa	-	Aleatória	10%	10%	-	10%	10%
ILS12	Gulosa	-	Sequencial	10%	-	10%	-	10%		ILS69	Gulosa	-	Aleatória	10%	10%	-	-	10%
ILS13	Gulosa	-	Sequencial	10%	-	-	10%	-		ILS70	Gulosa	-	Aleatória	10%	-	10%	-	-
ILS14	Gulosa	-	Sequencial	10%	-	-	10%	10%		ILS71	Gulosa	-	Aleatória	10%	-	10%	10%	-
ILS15	Gulosa	-	Sequencial	10%	-	-	-	10%		ILS72	Gulosa	-	Aleatória	10%	-	10%	10%	10%
ILS16	Gulosa	-	Sequencial	-	10%	-	-	-		ILS73	Gulosa	-	Aleatória	10%	-	10%	-	10%
ILS17	Gulosa	-	Sequencial	-	10%	10%	-	-		ILS74	Gulosa	-	Aleatória	10%	-	-	10%	-
ILS18	Gulosa	-	Sequencial	-	10%	10%	10%	-		ILS75	Gulosa	-	Aleatória	10%	-	-	10%	10%
ILS19	Gulosa	-	Sequencial	-	10%	10%	10%	10%		ILS76	Gulosa	-	Aleatória	10%	-	-	-	10%
ILS20	Gulosa	-	Sequencial	-	10%	10%	-	10%		ILS77	Gulosa	-	Aleatória	-	10%	10%	-	-
ILS21	Gulosa	-	Sequencial	-	10%	-	10%	-		ILS78	Gulosa	-	Aleatória	-	10%	10%	10%	-
ILS22	Gulosa	-	Sequencial	-	10%	-	10%	10%		ILS79	Gulosa	-	Aleatória	-	10%	10%	10%	10%
ILS23	Gulosa	-	Sequencial	-	10%	-	-	10%		ILS80	Gulosa	-	Aleatória	-	10%	10%	-	10%
ILS24	Gulosa	-	Sequencial	-	-	10%	-	-		ILS81	Gulosa	-	Aleatória	-	10%	-	10%	-
ILS25	Gulosa	-	Sequencial	-	-	10%	10%	-		ILS82	Gulosa	-	Aleatória	-	10%	-	10%	10%
ILS26	Gulosa	-	Sequencial	-	-	10%	10%	10%		ILS83	Gulosa	-	Aleatória	-	10%	-	-	10%
ILS27	Gulosa	-	Sequencial	-	-	10%	-	10%		ILS84	Gulosa	-	Aleatória	-	-	10%	10%	-
ILS28	Gulosa	-	Sequencial	-	-	-	10%	-		ILS85	Gulosa	-	Aleatória	-	-	10%	10%	10%
ILS29	Gulosa	-	Sequencial	-	-	-	10%	10%		ILS86	Gulosa	-	Aleatória	-	-	10%	-	10%
ILS30	Gulosa	-	Sequencial	-	-	-	-	10%		ILS87	Gulosa	-	Aleatória	-	-	-	10%	10%
ILS31	Gulosa	-	Sequencial	10%	10%	10%	10%	10%		ILS88	Gulosa	-	Aleatória	10%	10%	10%	10%	10%
ILS32	Gulosa	Sim	Sequencial	10%	-	-	-	-		ILS89	Gulosa	Sim	Aleatória	10%	10%	-	-	-
ILS33	Gulosa	Sim	Sequencial	10%	10%	-	-	-		ILS90	Gulosa	Sim	Aleatória	10%	10%	10%	10%	-
ILS34	Gulosa	Sim	Sequencial	10%	10%	10%	-	-		ILS91	Gulosa	Sim	Aleatória	10%	10%	10%	10%	-
ILS35	Gulosa	Sim	Sequencial	10%	10%	10%	10%	-		ILS92	Gulosa	Sim	Aleatória	10%	10%	10%	-	10%
ILS36	Gulosa	Sim	Sequencial	10%	10%	10%	-	10%		ILS93	Gulosa	Sim	Aleatória	10%	10%	-	10%	-
ILS37	Gulosa	Sim	Sequencial	10%	10%	-	10%	-		ILS94	Gulosa	Sim	Aleatória	10%	10%	-	10%	10%
ILS38	Gulosa	Sim	Sequencial	10%	10%	-	10%	10%		ILS95	Gulosa	Sim	Aleatória	10%	10%	-	-	10%
ILS39	Gulosa	Sim	Sequencial	10%	10%	-	-	10%		ILS96	Gulosa	Sim	Aleatória	10%	-	10%	-	-
ILS40	Gulosa	Sim	Sequencial	10%	-	10%	-	-		ILS97	Gulosa	Sim	Aleatória	10%	-	10%	10%	-
ILS41	Gulosa	Sim	Sequencial	10%	-	10%	10%	-		ILS98	Gulosa	Sim	Aleatória	10%	-	10%	10%	10%
ILS42	Gulosa	Sim	Sequencial	10%	-	10%	10%	10%		ILS99	Gulosa	Sim	Aleatória	10%	-	10%	-	10%
ILS43	Gulosa	Sim	Sequencial	10%	-	10%	-	10%		ILS100	Gulosa	Sim	Aleatória	10%	-	-	10%	-
ILS44	Gulosa	Sim	Sequencial	10%	-	-	10%	-		ILS101	Gulosa	Sim	Aleatória	10%	-	-	10%	10%
ILS45	Gulosa	Sim	Sequencial	10%	-	-	10%	10%		ILS102	Gulosa	Sim	Aleatória	10%	-	-	-	10%
ILS46	Gulosa	Sim	Sequencial	10%	-	-	-	10%		ILS103	Gulosa	Sim	Aleatória	-	10%	10%	-	-
ILS47	Gulosa	Sim	Sequencial	-	10%	-	-	-		ILS104	Gulosa	Sim	Aleatória	-	10%	10%	10%	-
ILS48	Gulosa	Sim	Sequencial	-	10%	10%	-	-		ILS105	Gulosa	Sim	Aleatória	-	10%	10%	10%	10%
ILS49	Gulosa	Sim	Sequencial	-	10%	10%	10%	-		ILS106	Gulosa	Sim	Aleatória	-	10%	10%	-	10%
ILS50	Gulosa	Sim	Sequencial	-	10%	10%	10%	10%		ILS107	Gulosa	Sim	Aleatória	-	10%	-	10%	-
ILS51	Gulosa	Sim	Sequencial	-	10%	10%	-	10%		ILS108	Gulosa	Sim	Aleatória	-	10%	-	10%	10%
ILS52	Gulosa	Sim	Sequencial	-	10%	-	10%	-		ILS109	Gulosa	Sim	Aleatória	-	10%	-	-	10%
ILS53	Gulosa	Sim	Sequencial	-	10%	-	10%	10%		ILS110	Gulosa	Sim	Aleatória	-	-	10%	10%	-
ILS54	Gulosa	Sim	Sequencial	-	10%	-	-	10%		ILS111	Gulosa	Sim	Aleatória	-	-	10%	10%	10%
ILS55	Gulosa	Sim	Sequencial	-	-	10%	-	-		ILS112	Gulosa	Sim	Aleatória	-	-	10%	-	10%
ILS56	Gulosa	Sim	Sequencial	-	-	10%	10%	-		ILS113	Gulosa	Sim	Aleatória	-	-	-	10%	10%
ILS57	Gulosa	Sim	Sequencial	-	-	10%	10%	10%		ILS114	Gulosa	Sim	Aleatória	10%	10%	10%	10%	10%

ID	Inicial	Mem.	Perturbação	Moves	Swaps	Splits	Joins	Exp.		ID	Inicial	Mem.	Perturbação	Moves	Swaps	Splits	Joins	Exp.
ILS115	Aleatória	-	Sequencial	10%	-	-	-	-		ILS172	Aleatório Sim	Sequencial	-	-	10%	-	10%	
ILS116	Aleatória	-	Sequencial	10%	10%	-	-	-		ILS173	Aleatório Sim	Sequencial	-	-	-	-	10%	
ILS117	Aleatória	-	Sequencial	10%	10%	10%	-	-		ILS174	Aleatório Sim	Sequencial	-	-	-	-	10%	
ILS118	Aleatória	-	Sequencial	10%	10%	10%	10%	-		ILS175	Aleatório Sim	Sequencial	-	-	-	-	10%	
ILS119	Aleatória	-	Sequencial	10%	10%	10%	-	10%		ILS176	Aleatório Sim	Sequencial	10%	10%	10%	10%	10%	
ILS120	Aleatória	-	Sequencial	10%	10%	-	10%	-		ILS177	Aleatório	Aleatória	10%	10%	-	-	-	
ILS121	Aleatória	-	Sequencial	10%	10%	-	10%	10%		ILS178	Aleatório	Aleatória	10%	10%	10%	-	-	
ILS122	Aleatória	-	Sequencial	10%	10%	-	-	10%		ILS179	Aleatório	Aleatória	10%	10%	10%	10%	-	
ILS123	Aleatória	-	Sequencial	10%	-	10%	-	-		ILS180	Aleatório	Aleatória	10%	10%	10%	-	10%	
ILS124	Aleatória	-	Sequencial	10%	-	10%	10%	-		ILS181	Aleatório	Aleatória	10%	10%	-	10%	-	
ILS125	Aleatória	-	Sequencial	10%	-	10%	10%	10%		ILS182	Aleatório	Aleatória	10%	10%	-	10%	10%	
ILS126	Aleatória	-	Sequencial	10%	-	10%	-	10%		ILS183	Aleatório	Aleatória	10%	10%	-	-	10%	
ILS127	Aleatória	-	Sequencial	10%	-	-	10%	-		ILS184	Aleatório	Aleatória	10%	-	-	10%	-	
ILS128	Aleatória	-	Sequencial	10%	-	-	10%	10%		ILS185	Aleatório	Aleatória	10%	-	-	10%	10%	
ILS129	Aleatória	-	Sequencial	10%	-	-	-	10%		ILS186	Aleatório	Aleatória	10%	-	-	10%	10%	
ILS130	Aleatória	-	Sequencial	-	10%	-	-	-		ILS187	Aleatório	Aleatória	10%	-	-	10%	-	
ILS131	Aleatória	-	Sequencial	-	10%	10%	-	-		ILS188	Aleatório	Aleatória	10%	-	-	-	10%	
ILS132	Aleatória	-	Sequencial	-	10%	10%	10%	-		ILS189	Aleatório	Aleatória	10%	-	-	-	10%	
ILS133	Aleatória	-	Sequencial	-	10%	10%	10%	10%		ILS190	Aleatório	Aleatória	10%	-	-	-	10%	
ILS134	Aleatória	-	Sequencial	-	10%	10%	-	10%		ILS191	Aleatório	Aleatória	-	10%	10%	-	-	
ILS135	Aleatória	-	Sequencial	-	10%	-	10%	-		ILS192	Aleatório	Aleatória	-	10%	10%	10%	-	
ILS136	Aleatória	-	Sequencial	-	10%	-	10%	10%		ILS193	Aleatório	Aleatória	-	10%	10%	10%	10%	
ILS137	Aleatória	-	Sequencial	-	10%	-	-	10%		ILS194	Aleatório	Aleatória	-	10%	10%	-	10%	
ILS138	Aleatória	-	Sequencial	-	-	10%	-	-		ILS195	Aleatório	Aleatória	-	10%	-	-	10%	
ILS139	Aleatória	-	Sequencial	-	-	10%	10%	-		ILS196	Aleatório	Aleatória	-	10%	-	-	10%	
ILS140	Aleatória	-	Sequencial	-	-	10%	10%	10%		ILS197	Aleatório	Aleatória	-	10%	-	-	10%	
ILS141	Aleatória	-	Sequencial	-	-	10%	-	10%		ILS198	Aleatório	Aleatória	-	-	-	10%	10%	
ILS142	Aleatória	-	Sequencial	-	-	-	10%	-		ILS199	Aleatório	Aleatória	-	-	-	10%	10%	
ILS143	Aleatória	-	Sequencial	-	-	-	10%	10%		ILS200	Aleatório	Aleatória	-	-	-	10%	-	
ILS144	Aleatória	-	Sequencial	-	-	-	-	10%		ILS201	Aleatório	Aleatória	-	-	-	-	10%	
ILS145	Aleatória	-	Sequencial	10%	10%	10%	10%	10%		ILS202	Aleatório	Aleatória	10%	10%	10%	10%	10%	
ILS146	Aleatória Sim	Sequencial	10%	-	-	-	-	-		ILS203	Aleatório Sim	Aleatória	10%	10%	-	-	-	
ILS147	Aleatória Sim	Sequencial	10%	10%	-	-	-	-		ILS204	Aleatório Sim	Aleatória	10%	10%	10%	-	-	
ILS148	Aleatória Sim	Sequencial	10%	10%	10%	-	-	-		ILS205	Aleatório Sim	Aleatória	10%	10%	10%	10%	-	
ILS149	Aleatória Sim	Sequencial	10%	10%	10%	10%	-	-		ILS206	Aleatório Sim	Aleatória	10%	10%	10%	-	10%	
ILS150	Aleatória Sim	Sequencial	10%	10%	10%	-	10%	-		ILS207	Aleatório Sim	Aleatória	10%	10%	-	-	10%	
ILS151	Aleatória Sim	Sequencial	10%	10%	-	-	10%	-		ILS208	Aleatório Sim	Aleatória	10%	10%	-	-	10%	
ILS152	Aleatória Sim	Sequencial	10%	10%	-	10%	10%	-		ILS209	Aleatório Sim	Aleatória	10%	10%	-	-	10%	
ILS153	Aleatória Sim	Sequencial	10%	10%	-	-	10%	-		ILS210	Aleatório Sim	Aleatória	10%	-	-	10%	-	
ILS154	Aleatória Sim	Sequencial	10%	-	10%	-	-	-		ILS211	Aleatório Sim	Aleatória	10%	-	-	10%	10%	
ILS155	Aleatória Sim	Sequencial	10%	-	10%	10%	-	-		ILS212	Aleatório Sim	Aleatória	10%	-	-	10%	10%	
ILS156	Aleatória Sim	Sequencial	10%	-	10%	10%	10%	-		ILS213	Aleatório Sim	Aleatória	10%	-	-	10%	10%	
ILS157	Aleatória Sim	Sequencial	10%	-	10%	-	10%	10%		ILS214	Aleatório Sim	Aleatória	10%	-	-	-	10%	
ILS158	Aleatória Sim	Sequencial	10%	-	-	10%	-	-		ILS215	Aleatório Sim	Aleatória	10%	-	-	-	10%	
ILS159	Aleatória Sim	Sequencial	10%	-	-	10%	10%	-		ILS216	Aleatório Sim	Aleatória	10%	-	-	-	10%	
ILS160	Aleatória Sim	Sequencial	10%	-	-	-	10%	-		ILS217	Aleatório Sim	Aleatória	-	10%	10%	-	-	
ILS161	Aleatória Sim	Sequencial	-	10%	-	-	-	-		ILS218	Aleatório Sim	Aleatória	-	10%	10%	10%	-	
ILS162	Aleatória Sim	Sequencial	-	10%	10%	-	-	-		ILS219	Aleatório Sim	Aleatória	-	10%	10%	10%	10%	
ILS163	Aleatória Sim	Sequencial	-	10%	10%	10%	-	-		ILS220	Aleatório Sim	Aleatória	-	10%	10%	-	10%	
ILS164	Aleatória Sim	Sequencial	-	10%	10%	10%	10%	-		ILS221	Aleatório Sim	Aleatória	-	10%	-	-	10%	
ILS165	Aleatória Sim	Sequencial	-	10%	10%	-	10%	-		ILS222	Aleatório Sim	Aleatória	-	10%	-	-	10%	
ILS166	Aleatória Sim	Sequencial	-	10%	-	-	10%	-		ILS223	Aleatório Sim	Aleatória	-	10%	-	-	10%	
ILS167	Aleatória Sim	Sequencial	-	10%	-	10%	10%	-		ILS224	Aleatório Sim	Aleatória	-	-	-	10%	10%	
ILS168	Aleatória Sim	Sequencial	-	10%	-	-	10%	-		ILS225	Aleatório Sim	Aleatória	-	-	-	10%	10%	
ILS169	Aleatória Sim	Sequencial	-	-	10%	-	-	-		ILS226	Aleatório Sim	Aleatória	-	-	-	10%	-	
ILS170	Aleatória Sim	Sequencial	-	-	10%	10%	-	-		ILS227	Aleatório Sim	Aleatória	-	-	-	-	10%	
ILS171	Aleatória Sim	Sequencial	-	-	10%	10%	10%	-		ILS228	Aleatório Sim	Aleatória	10%	10%	10%	10%	10%	

Anexo II – P-values e Effect-size para 9 configurações do ILS

INSTÂNCIAS	ILS1 > ILS3		ILS1 > ILS6		ILS1 & ILS63		ILS1 > ILS2		ILS1 > ILS64		ILS1 > ILS8		ILS1 > ILS77		ILS1 > ILS80	
	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE
nanoxml	1.000	50%	0.161	47%	0.161	47%	0.161	47%	0.654	50%	0.334	52%	0.022	50%	0.011	60%
apache_zip	0.452	53%	0.161	47%	0.959	50%	0.295	54%	0.435	52%	0.082	55%	0.001	65%	<0.001	73%
jsccatterplot	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%
junif	<0.001	93%	0.684	53%	0.217	59%	<0.001	80%	0.918	51%	<0.001	89%	0.641	46%	0.038	66%
gae_plugin_core	<0.001	8%	0.435	44%	0.833	48%	0.676	53%	<0.001	16%	<0.001	17%	<0.001	11%	<0.001	8%
pdndogram	0.797	52%	0.881	49%	0.858	51%	0.770	49%	0.587	49%	0.532	55%	0.587	54%	0.124	60%
pdf_renderer	0.000	77%	0.208	60%	0.208	60%	0.041	65%	0.941	49%	0.016	68%	0.463	44%	0.166	40%
lung_visualization	0.088	63%	0.015	68%	0.318	58%	0.164	61%	0.033	69%	0.033	66%	0.046	65%	0.007	70%
picda_swing	<0.001	91%	0.004	71%	0.115	62%	<0.001	88%	0.005	71%	<0.001	93%	0.039	68%	0.002	73%
jmt-1.0b4	<0.001	79%	0.483	55%	0.483	45%	0.021	67%	0.007	67%	0.004	71%	0.246	41%	0.525	45%
notelab-full	<0.001	82%	0.365	50%	0.356	45%	<0.001	83%	0.116	62%	<0.001	86%	0.213	59%	0.036	66%
Wilcoxon teste Effect-size do ILS1																
ILS16 < ILS6																
INSTÂNCIAS	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE
nanoxml	0.161	53%	0.161	53%	0.161	53%	0.022	50%	0.081	59%	0.334	52%	0.022	50%	0.011	60%
apache_zip	0.042	57%	0.161	53%	0.161	53%	0.022	50%	0.042	57%	0.082	55%	0.001	65%	<0.001	73%
jsccatterplot	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%
junif	<0.001	90%	0.389	56%	0.684	47%	<0.001	78%	0.888	49%	<0.001	89%	0.728	47%	0.096	63%
gae_plugin_core	<0.001	13%	0.510	55%	0.435	51%	0.327	57%	0.001	25%	<0.001	23%	<0.001	18%	<0.001	14%
pdndogram	0.428	56%	0.477	55%	0.881	51%	0.842	52%	0.822	52%	0.195	59%	0.214	59%	0.045	64%
pdf_renderer	0.093	63%	0.662	47%	0.208	40%	0.647	54%	0.113	38%	0.318	56%	0.024	33%	0.003	28%
lung_visualization	0.552	45%	0.343	43%	0.015	32%	0.096	37%	0.230	41%	0.809	48%	0.971	50%	0.197	60%
picda_swing	0.001	76%	0.234	41%	0.004	29%	0.002	73%	0.807	48%	<0.001	77%	0.408	44%	0.813	52%
jmt-1.0b4	0.001	75%	0.181	40%	0.483	45%	0.064	64%	0.085	37%	0.017	68%	0.058	36%	0.197	40%
notelab-full	<0.001	80%	0.356	43%	0.985	50%	<0.001	80%	0.268	58%	<0.001	82%	0.173	60%	0.042	65%
Wilcoxon teste Effect-size do ILS16																
ILS63 > ILS3																
INSTÂNCIAS	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE
nanoxml	0.161	53%	0.161	53%	0.161	53%	0.283	55%	0.081	59%	0.334	52%	0.022	50%	0.011	60%
apache_zip	0.371	54%	0.161	47%	0.959	50%	0.283	55%	0.356	54%	0.633	52%	0.016	62%	<0.001	71%
jsccatterplot	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%
junif	<0.001	90%	0.389	44%	0.217	41%	0.001	75%	0.308	42%	<0.001	88%	0.206	40%	0.290	58%
gae_plugin_core	<0.001	8%	0.510	45%	0.833	52%	0.582	54%	0.000	20%	<0.001	19%	<0.001	13%	<0.001	10%
pdndogram	0.884	51%	0.477	45%	0.858	49%	0.643	47%	0.645	47%	0.424	56%	0.599	54%	0.068	62%
pdf_renderer	0.001	76%	0.662	53%	0.208	40%	0.258	59%	0.156	39%	0.065	64%	0.018	32%	0.001	25%
lung_visualization	0.719	53%	0.343	43%	0.224	41%	0.684	47%	0.976	50%	0.328	57%	0.375	57%	0.097	63%
picda_swing	<0.001	83%	0.234	59%	0.115	36%	<0.001	79%	0.325	57%	<0.001	84%	0.647	54%	0.120	62%
jmt-1.0b4	<0.001	83%	0.181	40%	0.483	55%	0.004	72%	0.369	44%	0.000	76%	0.311	42%	0.923	49%
notelab-full	<0.001	84%	0.356	57%	0.356	57%	<0.001	83%	0.050	65%	<0.001	84%	0.033	66%	0.014	68%
Wilcoxon teste Effect-size do ILS63																
ILS3 < ILS16																
INSTÂNCIAS	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE	P-VALUE	EFF-SIZE
nanoxml	0.161	47%	0.161	47%	1.000	50%	0.161	47%	0.654	50%	0.570	48%	0.227	50%	0.129	57%
apache_zip	0.042	43%	0.371	46%	0.452	47%	0.821	51%	0.990	50%	0.740	49%	0.126	58%	0.005	67%
jsccatterplot	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%	-	50%
junif	<0.001	10%	<0.001	10%	<0.001	7%	0.095	34%	<0.001	9%	0.424	56%	<0.001	15%	<0.001	19%
gae_plugin_core	<0.001	87%	<0.001	92%	<0.001	92%	0.000	96%	0.003	71%	0.020	67%	0.142	60%	0.730	52%
pdndogram	0.428	44%	0.884	49%	0.797	48%	0.522	45%	0.579	45%	0.573	54%	0.703	53%	0.129	61%
pdf_renderer	0.093	37%	0.001	24%	<0.001	23%	0.101	38%	<0.001	18%	0.544	45%	<0.001	16%	<0.001	11%
lung_visualization	0.552	55%	0.719	47%	0.088	37%	0.412	44%	0.633	46%	0.825	52%	0.613	54%	0.187	60%
picda_swing	0.001	24%	<0.001	16%	0.775	9%	0.132	39%	<0.001	23%	0.804	52%	<0.001	18%	0.002	27%
jmt-1.0b4	0.001	25%	<0.001	17%	<0.001	21%	0.132	39%	<0.001	17%	0.282	42%	<0.001	18%	<0.001	19%
notelab-full	<0.001	20%	<0.001	16%	<0.001	18%	0.796	48%	0.001	25%	0.877	49%	0.008	30%	0.048	35%
Wilcoxon teste Effect-size do ILS3																

Anexo III - *P-values e Effect-size GNE (one-point crossover x two-point-crossover)*

INSTÂNCIAS	GNE14 > GNE7		GNE13 > GNE6		GNE12 > GNE5		GNE11 > GNE4	
	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>
JNANOXML	0,199	59%	0,181	40%	0,502	55%	0,675	47%
APACHE	0,278	58%	0,411	56%	0,594	54%	0,653	47%
JSCATTERPLOT	0,457	44%	0,214	59%	0,062	64%	0,760	48%
JUNIT	0,233	59%	0,200	60%	0,699	53%	0,164	60%
TINYTIM	0,273	58%	0,343	57%	0,168	60%	0,764	52%
GAE_PLUGIN	<0,001	96%	0,784	48%	<0,001	86%	0,420	44%
JDENDOGRAM	<0,001	90%	0,003	72%	<0,001	90%	<0,001	86%
PDF_RENDERER	<0,001	89%	<0,001	82%	0,001	75%	0,010	70%
JUNG_VISUALIZATION	<0,001	92%	<0,001	87%	<0,001	93%	<0,001	92%
PFCDA_SWING	<0,001	100%	<0,001	99%	<0,001	99%	<0,001	98%
JML	<0,001	99%	<0,001	93%	<0,001	97%	<0,001	95%
NOTEPAD_FULL	<0,001	94%	<0,001	100%	<0,001	97%	<0,001	99%

INSTÂNCIAS	GNE10 > GNE3		GNE9 > GNE2		GNE8 > GNE1	
	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>	<i>P-value</i>	<i>Effect-Size</i>
JNANOXML	0,413	56%	0,396	56%	0,428	56%
APACHE	0,625	54%	0,944	49%	0,483	54%
JSCATTERPLOT	0,847	49%	0,640	54%	<0,001	93%
JUNIT	0,477	55%	0,722	53%	<0,001	89%
TINYTIM	0,100	62%	0,177	40%	<0,001	96%
GAE_PLUGIN	0,178	60%	<0,001	89%	<0,001	97%
JDENDOGRAM	<0,001	79%	<0,001	81%	<0,001	96%
PDF_RENDERER	<0,001	85%	<0,001	78%	<0,001	99%
JUNG_VISUALIZATION	<0,001	84%	<0,001	91%	<0,001	94%
PFCDA_SWING	<0,001	97%	<0,001	93%	<0,001	97%
JML	<0,001	97%	<0,001	91%	<0,001	82%
NOTEPAD_FULL	<0,001	90%	<0,001	96%	<0,001	97%