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

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS APLICADO AO PROBLEMA DE CLUSTERIZAÇÃO DE MÓDULOS DE SOFTWARE

Geraldo Luiz B. Megale

Orientador
Adriana C. de F. Alvim

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2015

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS APLICADO AO PROBLEMA DE CLUSTERIZAÇÃO DE MÓDULOS DE SOFTWARE

Geraldo Luiz B. Megale

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

Aprovada por:

Prof. Adriana C. de F. Alvim, D.Sc – UNIRIO

Prof. Marcio de Oliveira Barros, D.Sc – UNIRIO

Prof. Isabel Cristina Mello Rosseti, D.Sc – UFF

RIO DE JANEIRO, RJ – BRASIL

Outubro de 2015

Megale, Geraldo Luiz B.
M496 Algoritmo genético de chaves aleatórias viciadas aplicado ao problema de clusterização de módulos de software / Geraldo Luiz B. Megale, 2015.
xiii, 87 f. ; 30 cm

Orientadora: Adriana C. de F. Alvim.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2015.

1. Engenharia de software. 2. Algoritmos genéticos. 3. Ferramentas de busca. 4. Clusterização de módulos de software. I. Alvim, Adriana C. de F. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III. Título.

CDD – 005.1

A minha mãe, a meu irmão e a meu pai (in memorian)

Agradecimentos

Agradeço primeiramente à meus pais e meus avós, que tanto se sacrificaram para que eu tivesse a melhor educação possível e continuam sendo essenciais na minha vida acadêmica, pessoal e profissional.

Agradeço a meu irmão, Ivan Lopes Jr, pelo incentivo a investir na área de ciência da computação.

Agradeço a minha orientadora Adriana Alvim, que além da orientação exemplar e tremenda paciência, nunca esteve ausente nos momentos necessários.

Agradeço aos membros da banca, professora Isabel Cristina Mello Rosseti e ao professor Marcio de Oliveira Barros, pelas sugestões que muito contribuíram para melhorar a qualidade final do texto da dissertação.

Agradeço aos meus colegas de trabalho, em especial meus chefes Roberto Kothe Werlang e Jose Maurício Mazin que, sem o apoio e incentivo, este mestrado não seria concretizado.

Agradeço ao amigo Guilherme Ferreira Lima pelo auxílio nas partes mais complexas das heurísticas.

Agradeço a minha noiva Marcela Rocha sempre companheira e essencial para me manter tranquilo e focado nos meus objetivos.

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.

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. (Melvyn Conway)

Geraldo Luiz B. Megale. **Algoritmo Genético de Chaves Aleatórias Viciadas Aplicado ao Problema de Clusterização de Módulos de Software**. 87 páginas. Dissertação de Mestrado. Programa de Pós-Graduação em Informática, UNIRIO.

Resumo

Um software é composto por um conjunto de arquivos que podem ser agrupados de muitas maneiras distintas. Esse processo de agrupamento é denominado de clusterização de módulos de software. O principal objetivo desse agrupamento é facilitar o entendimento de um software e, consequentemente, simplificar suas futuras manutenções e evoluções. A principal contribuição deste trabalho de dissertação de mestrado consiste no desenvolvimento de uma heurística, baseada no método algoritmos genéticos de chaves aleatórias viciadas (BRKGA), para resolver o problema de clusterização de módulos de software. A heurística proposta utiliza os seguintes componentes: algoritmo de codificação da solução, algoritmo de decodificação da solução e busca local. Inicialmente, considerando diversas escolhas possíveis para os parâmetros do framework BRKGA e para os componentes da heurística, 13 experimentos foram realizados a fim de eleger a melhor configuração para o algoritmo proposto, totalizando 9.120 execuções. Em seguida, comparou-se os resultados obtidos pela melhor versão do algoritmo proposto com seis métodos da literatura (quatro algoritmos genéticos, um método exato e uma heurística baseada na metaheurística Iterated Local Search), de forma isolada e, por fim, comparou-se com os melhores resultados da literatura para um conjunto de 113 instâncias. Resultados computacionais indicam que, quando comparada com outros métodos baseados em algoritmos genéticos, a heurística proposta é capaz de obter, na maioria dos casos, soluções de melhor qualidade. Entretanto, esta melhoria é conseguida com um custo computacional elevado, se comparado ao dos demais métodos. Outra contribuição deste trabalho foi a implementação do framework BRKGA na linguagem C#.NET.

Palavras-chaves: clusterização de módulos de software, algoritmos genéticos de chaves aleatórias viciadas, engenharia de software baseada em busca.

Abstract

A Software is made of several files that can be grouped in many different ways. This grouping process is called Software Module Clustering. Its main objective is to make it easier to understand a software thus facilitating its future maintenance and evolution. This masters' dissertation main contribution is the development of a heuristic based on the Biased Random Key Genetic Algorithm (BKRGA) to solve the Software Module Clustering Problem. The proposed heuristic uses the following components: a coding algorithm, decoding algorithm and a local search. At first, given many possible choices for the BRKGA parameters and its components, 13 experiments were executed in order to find the best configuration for the proposed algorithm in a total of 9.120 executions. The next step used the best configuration of the proposed algorithm discovered on the previous experiment with 6 algorithms obtained from the literature (four genetic algorithms, one exact algorithm and one heuristic based on the metaheuristic Iterated local Search). At last, the best results taken from the literature were compared for 113 instances. The results indicate that, when compared to other genetic algorithms, the proposed heuristic is able to obtain, in most cases, better quality solutions. However, this improvement is obtained at a high computational cost if compared to other algorithms. Another contribution of this dissertation is the implementation of the BRKGA framework in C#.NET.

Key-words: Software Module Clustering, Biased Random Keys Genetic Algorithm, Search Based Software Engineering

Sumário

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

ATMR	Attributes to methods ratio
AG	Algoritmo Genético
AGI	Algoritmo Genético Iterativo
BL	Busca Local
BRKGA	Biased Random Keys Genetic Algorithm
ECA	Equal Size Approach
EC	External Couple Elegance
GGA	Grouping Genetic Algorithm
GNE	Grouping Number Encoding
ILS	Iterated Local Search
ILS	Internal uses elegance
MQ	Modularization Quality
MCA	Maximizing Cluster Approach
MDG	Module Dependency Graph
MF	Modularization Factor
NAC	Number Among Class
NSGA-II	Non Dominating Sorting Genetic Algorithm
PCMS	Problema da Clusterização de Módulos de Software
RKGA	Random Keys Genetic Algorithm
RNC	Radio Network Controller

1 Introdução

A manutenção, tanto evolutiva quanto corretiva, de sistemas de software são atividades complexas e demoradas especialmente se a documentação desses sistemas estiverem desatualizadas ou simplesmente inexistirem (DOVAL; MANCORIDIS; MITCHELL, 1999). Mesmo sistemas de softwares robustos têm uma alta probabilidade de terem sua complexidade aumentada de forma não linear (BROOKS JR., 1987). Além disso, é bastante comum atividades ligadas a gestão do conhecimento não serem priorizadas em detrimento da construção do software em si, o que ocasiona um problema futuro quando este necessitar de alguma evolução ou manutenção, já que o conhecimento necessário para realizar essas atividades de forma eficiente, sem impactos retroativos, perdeu-se ao longo da construção do mesmo. A falta de gestão do conhecimento e a consequente dificuldade de realizar uma manutenção ou evolução é ainda agravada pela alta rotatividade de profissionais técnicos, algo bastante comum na área de Engenharia de Software (MANCORIDIS et al., 1999).

De forma a mitigar a dificuldade de evolução/manutenção, arquitetos de software utilizam grafos direcionados para facilitar o entendimento da complexa estrutura de um sistema de software. Esses grafos são chamados de grafos de dependências de módulos, em inglês *Module Dependency Graphs* (MDG), e representam os módulos de um sistema (classes, arquivos) e os seus relacionamentos (chamadas de função, herança, implementação de interfaces, etc.) como arestas direcionadas conectando os nós do grafo (DOVAL; MANCORIDIS; MITCHELL, 1999). Outra maneira para facilitar ainda mais o entendimento da complexa estrutura de um sistema de software é particionar os nós do MDG em estruturas lógicas agrupando os nós (módulos) com base em algum critério de afinidade/similaridade. Esse processo de agrupamento é denominado de clusterização de módulos de software e tem como objetivo criar grupos de módulos (*clusters*) garantindo um fácil entendimento do sistema de software.

Uma boa clusterização garante inúmeras facilidades de desenvolvimento, testabilidade, manutenção, evolução do software e facilidades ao projetar um sistema de software (CONSTANTINE; YOURDON, 1979), impactando positivamente na sua qualidade total e consequentemente na sua robustez.

De forma simplificada, o problema de clusterização de módulos de software (PCMS) consiste em, dado um conjunto de módulos e suas relações de dependência, particionar os módulos em conjuntos disjuntos (*clusters*) objetivando maximizar algum critério de qualidade. Esse problema pode ser visto como um problema de clusterização, ou de particionamento de grafos (em que vértices representam os módulos e as arestas representam os relacionamentos), que é NP-Difícil (GAREY; JOHNSON, 1979). Para tal classe de problemas, o uso de heurísticas é capaz de fornecer soluções de qualidade em tempo razoável.

Na literatura, existem diversas abordagens para estimar a qualidade de uma clusterização. Esse trabalho utiliza como critério para avaliar uma boa clusterização os conceitos de coesão (medida de quanto um módulo está dedicado para resolver um problema específico) e acoplamento (medida do grau de dependência de um módulo em relação aos demais) que são computados a partir de uma única função objetivo: a função qualidade da modularização, em inglês *Modularization Quality* (MQ) (DOVAL; MANCORIDIS; MITCHELL, 1999).

Metaheurísticas são procedimentos de alto nível que coordenam heurísticas simples, como, por exemplo, buscas locais. Em geral, metaheurísticas encontram soluções de melhor qualidade quando comparadas com aquelas encontradas, de forma isolada, por heurísticas simples (GONÇALVES; RESENDE, 2011).

O presente trabalho de dissertação de mestrado tem como objetivo encontrar de forma automática soluções de qualidade para o PCMS por meio da implementação de uma heurística baseada na metaheurística algoritmos genéticos de chaves aleatórias viciadas, em inglês *Biased Random-key Genetic Algorithms* (BRKGA) (GONÇALVES; RESENDE, 2011). Recentemente, na literatura, a técnica BRKGA têm sido empregada com sucesso para resolver problemas de natureza similar a do PCMS, como pode ser visto em (GONÇALVES; RESENDE, 2011) e em (MORÁN-MIRABAL et al., 2013).

Uma das motivações para se eleger o BRKGA como método de resolução é que, até o presente, não se conhece literatura que use a técnica BRKGA para tratar o PCMS, acreditando que a investigação proposta seja uma contribuição importante para as Áreas de Engenharia de Software e de Heurísticas.

1.1 Definição do problema

Conforme Hansen e Jaumard (HANSEN; JAUMARD, 1997) problemas de clusterização consideram um conjunto de entidades juntamente com algumas medidas que descrevem essas entidades. O objetivo é encontrar subconjuntos de interesse, chamados de *clusters*, dentro do conjunto maior de entidades. Usualmente *clusters* devem ser homogêneos e/ou bem separados. Homogeneidade significa que as entidades dentro de um mesmo *cluster* devem se parecer umas com as outras e separação significa que as entidades em *clusters* distintos devem ser diferentes umas das outras. Esse problema é antigo e aparece em diversas áreas do conhecimento. Ainda, segundo Hansen e Jaumard (HANSEN; JAUMARD, 1997), os problemas de clusterização podem ser classificados nos seguintes tipos: de subconjunto, de particionamento, de empacotamento, de recobrimento e hierárquico.

O problema de clusterização de módulos de software pode ser modelado como um problema de clusterização do tipo de particionamento, conforme mostrado a seguir. Considere $N = \{1, 2, \ldots, n\}$ o conjunto de n módulos que fazem parte de um sistema de software. O PCMS consiste em encontrar um particionamento $C = \{C_1, C_2, \ldots, C_k\}$ de N em k *clusters*,

com $C_i \neq \emptyset, i = 1, 2, \ldots, k$; $C_i \cap C_j = \emptyset, i, j = 1, 2, \ldots, k$ e $i \neq j$; e $\bigcup_{i=1}^{k} C_i = N$ de tal forma a maximizar/minimizar algum critério. Isto é, não existe *cluster* vazio, cada módulo pertence a exatamente um *cluster* e a união de todos os elementos dos *clusters* é igual ao conjunto de módulos N. Quando a quantidade de clusters k é um dado do problema (entrada), o problema é conhecido como problema de k-clusterização, e quando o número de *clusters* é uma saída do problema, ele é conhecido como o problema de clusterização automática (SEMAAN; OCHI, 2007). No caso do PCMS, o número ideal k de *clusters* não é conhecido a *priori* e deve ser encontrado pelo método de resolução do problema.

Em grafos de dependência de módulos (MDG), os módulos do sistema (por exemplo, arquivos e classes) são representados pelos nós e as relações entre os módulos (por exemplo, chamada de função e herança) são representadas por arestas direcionadas. Dado que um grafo MDG pode ser usado para representar a estrutura de um sistema de software, o PCMS também pode ser modelado como um problema de particionamento de grafos.

Formalmente, MDG $= G(V, E)$, onde $V = \{1, 2, \ldots, n\}$ é o conjunto de nós e $E = \{(u, v) | u, v \in V\}$ o conjunto de arestas ponderadas, tal que, para cada aresta $(u, v) \in E$ existe um peso positivo associado dado por w_{uv}. Os pesos nas arestas podem ser usados para representar diferentes tipos de relacionamento entre os módulos.

O objetivo do PCMS é encontrar uma partição de G como $P_G = G_1, G_2, \ldots, G_k$, em k $(1 \leq k \leq n)$ *clusters* de forma que cada G_i $(i = 1, \ldots, k)$ é um *cluster* no grafo particionado. A seguir, a formulação do problema conforme Mitchell (MITCHELL, 2002):

$$
\begin{gathered}
G = (V, E), P_G = \bigcup_{i=1}^{k} G_i \\
G_i = (V_i, E_i) \\
\bigcup_{i=1}^{k} V_i = V \\
V_i \cap V_j = \emptyset, \ \forall (i, j = 1, \ldots, k), i \neq j \\
E_i = \{(u, v) \in E | u \in V_i \wedge v \in V\}.
\end{gathered}
$$

Se P_G é o conjunto dos *clusters* do MDG, cada G_i possui um conjunto V_i de módulos de V, que não se sobrepõem, e um conjunto E_i de arestas de E. O número de *clusters* da partição varia entre 1 (um único *cluster* com todos os módulos) e $|V|$ (cada módulo em um *cluster* único).

Neste trabalho, a função objetivo usada para avaliar a qualidade de uma partição e que deverá ser maximizada é a qualidade da modularização na sua versão Turbo MQ (MITCHELL, 2002), descrita na Seção 2.1.2.

1.2 Objetivos

Existem diversas técnicas para resolução do PCMS incluindo buscas locais, algoritmos genéticos mono-objetivo e multiobjetivo e programação linear inteira mista (MANCORIDIS et al., 1999; PINTO, 2014; DOVAL; MANCORIDIS; MITCHELL, 1999; PRADITWONG, 2011; PRADITWONG; HARMAN; YAO, 2011; KÖHLER; FAMPA; ARAUJO, 2013). De acordo com o levantamento bibliográfico realizado nesta dissertação não existe utilização da metaheurística BRKGA (GONÇALVES; RESENDE, 2014) para o PCMS. Além disso, para diversos problemas complexos, inclusive de clusterização, a técnica BRKGA obteve resultados bastante satisfatórios em termos de eficácia.

Sendo assim, este levantamento motivou o presente trabalho de dissertação de mestrado, cujo objetivo geral é propor uma estratégia heurística baseada na metaheurística BRKGA para encontrar soluções de qualidade e em tempo comparável a outros algoritmos genéticos para o PCMS.

Para alcançar o objetivo geral é necessário uma pesquisa de caráter teórico compreendendo o estudo aprofundado do método de resolução do problema em questão e uma pesquisa de caráter experimental: analisar os dados produzidos pelos testes, tanto no que se refere à qualidade das soluções, especificidade das instâncias e tempos relativos. Desta forma, é possível enumerar os seguintes importantes objetivos específicos desta dissertação:

1. Desenvolver e implementar uma heurística para solucionar o problema clusterização de módulos de software baseada na metaheurística BRKGA.

2. Desenvolver um projeto rigoroso de experimentos (BARR et al., 1995) consistindo das seguintes etapas:

   a) Selecionar um conjunto expressivo de instâncias teste;

   b) Caracterizar e eleger a heurística proposta. Métodos baseados em algoritmos genéticos, em geral, possuem diversos parâmetros que devem ser configurados. Além disso, a metaheurística BRKGA possui alguns componentes livres (por exemplo, codificadores e decodificadores) que devem ser projetados para o problema em questão. O objetivo desta etapa é estudar e analisar diversas configurações dos parâmetros do BRKGA e propor diversas versões para os componentes livres do BRKGA, a fim de eleger a versão mais robusta como proposta de solução para o PCMS;

   c) Comparar, analisar e interpretar os resultados obtidos pelo método proposto com os resultados obtidos por quatro outros métodos baseados em algoritmos genéticos, um método baseado em *Iterated Local Search* e um método exato disponíveis na literatura, quanto à qualidade das soluções (eficácia) e tempos de processamento (eficiência); e

d) Comparar, analisar e interpretar os resultados obtidos pelo método proposto com os melhores resultados disponíveis na literatura, quanto à qualidade das soluções (eficácia) e tempos de processamento (eficiência).

1.3 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 (PCMS) e os objetivos da pesquisa. No Capítulo 2 é apresentada uma revisão bibliográfica dos trabalhos relacionados. O Capítulo 3 apresenta a aplicação da metaheurística BRKGA ao PCMS. No Capítulo 4 os experimentos e as comparações dos resultados obtidos nesta dissertação são detalhados. Por último, o Capítulo 5 apresenta as considerações finais, com as conclusões, contribuições da pesquisa e propostas de trabalhos futuros.

2 Revisão Bibliográfica

Este capítulo está dividido em três seções. A primeira seção apresenta algumas funções objetivos da literatura usadas para avaliar a qualidade de uma solução do PCMS. A segunda seção apresenta o *framework* algoritmos genéticos de chaves aleatórias viciadas (BRKGA), usado nesta dissertação para propor uma solução para o PCMS. A terceira e última seção faz um levantamento dos principais trabalhos, suas abordagens e resultados aplicados ao PCMS e problemas correlatos.

2.1 Medidas de qualidade para o PCMS

Nesta seção, inicialmente detalha-se três formas distintas de se calcular a função objetivo qualidade da modularização (MQ), conforme apresentado por Mitchell (MITCHELL, 2002). Em seguida, apresenta-se a métrica *Evaluation Metric* (EVM).

2.1.1 Função objetivo MQ Básico

A função objetivo chamada de MQ Básico, introduzida por Mancoridis et al. (MANCORIDIS et al., 1998) e utilizada em diversos trabalhos (MANCORIDIS et al., 1999; PINTO, 2014; DOVAL; MANCORIDIS; MITCHELL, 1999; PRADITWONG, 2011; PRADITWONG; HARMAN; YAO, 2011; KÖHLER; FAMPA; ARAUJO, 2013), mede a qualidade de uma dada clusterização. Para isso, considere as seguintes definições:

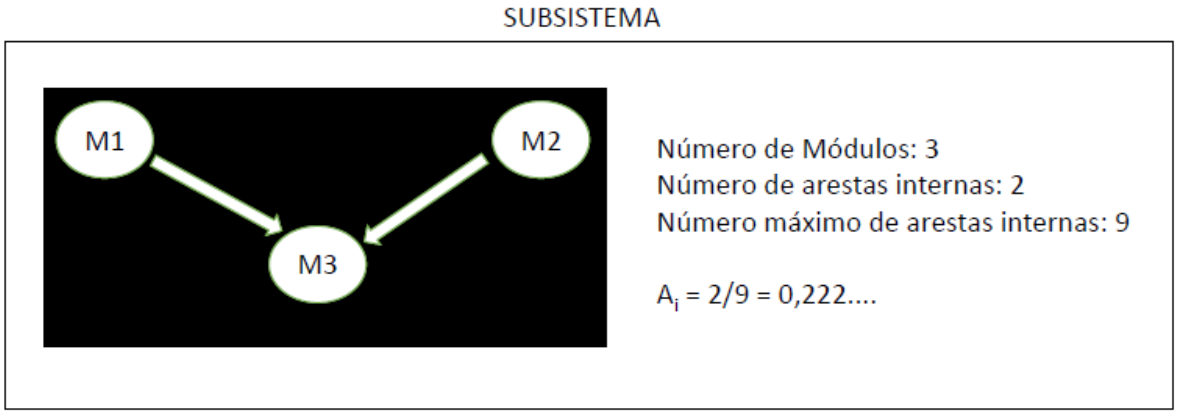
1. Intra-conectividade: é a medida da conectividade de módulos pertencentes a um mesmo *cluster*. Um alto grau de intra-conectividade indica uma boa clusterização pois os módulos agrupados no mesmo *cluster* compartilham entre si muitas de suas funcionalidades. Alternativamente, um baixo grau de intra-conectividade indica que apesar de estarem agrupados em uma mesma estrutura lógica, os módulos do *cluster* não compartilham entre si muitas de suas funcionalidades. Formalmente, seja A_i a intra-conectividade do *cluster* i que possui N_i componentes e μ_i dependências internas, tem-se:

$$A_i = \frac{\mu_i}{N_i^2} \tag{2.1}$$

   Essa medida é a fração do máximo de dependências internas que podem existir para o *cluster* i, no caso, N_i^2. É fácil perceber que A_i está limitado por valores entre 0 e 1 sendo que 0 significa que no *cluster* i os módulos não utilizam nenhuma funcionalidade dos outros módulos pertencentes ao mesmo *cluster* e 1 significa que todos os módulos do *cluster* utilizam pelo menos uma funcionalidade de todos os outros módulos do *cluster* (nesse

caso, tem-se um grafo completo). A Figura 1 exemplifica o cálculo de intra-conectividade de um *cluster*, na figura identificado como subsistema. Sejam $N_i = 3, \mu_i = 2$ e $N_i^2 = 9$, respectivamente, o número de módulos, o número de dependências internas e o quadrado do número de componentes do *cluster* i; a intra-conectividade é dada por $A_i = \frac{\mu_i}{N_i^2} = \frac{2}{9} = 0.2222\ldots$.

Figura 1: Exemplo de cálculo de intra-conectividade (MANCORIDIS et al., 1998).



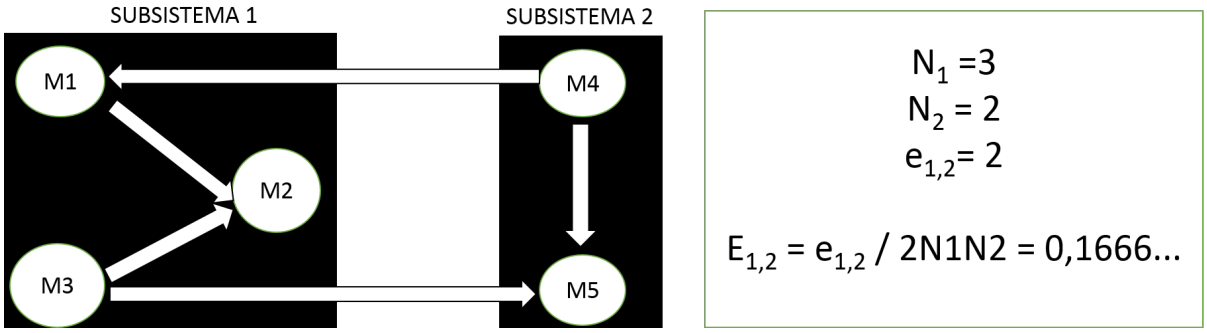
2. Inter-conectividade: é a medida da conectividade de módulos pertencentes a *clusters* distintos. Um alto grau de inter-conectividade é uma indicação de uma clusterização de baixa qualidade pois uma mudança em um dos módulos pode propagar mudanças em outras partes do sistema de software devido a esse alto grau de inter-conectividade. Alternativamente, um baixo grau de inter-conectividade indica que os *clusters* são bem independentes, fazendo com que mudanças em seus módulos não se propaguem além das fronteiras de seu *cluster*. Formalmente, seja E_{ij} a inter-conectividade entre os *clusters* i e j consistindo dos componentes N_i e N_j, respectivamente e ϵ_{ij} o total de dependências entre os *clusters* i e j.

$$E_{ij} = \begin{cases} 0 & \text{se } i = j \\ \frac{\varepsilon_{ij}}{2N_iN_j} & \text{se } i \neq j \end{cases} \tag{2.2}$$

A Figura 2 exemplifica o cálculo de inter-conectividade entre dois *clusters*, na figura identificados como subsistema 1 e subsistema 2. Sejam $N_1 = 3, N_2 = 2$ e $\epsilon_{1,2} = 2$, respectivamente, o número de módulos do *cluster* 1, o número de módulos do *cluster* 2 e o total de dependências entre os *clusters* 1 e 2; a inter-conectividade entre os *clusters* 1 e 2 é dada por $E_{1,2} = \frac{\epsilon_{1,2}}{2N_1N_2} = \frac{2}{12} = 0.1666\ldots$.

3. Qualidade da modularização: é a medida da modularização de um sistema de software. O MQ Básico de um grafo particionado em k *clusters*, onde A_i é a intra-conectividade do i-ésimo *cluster* e E_{ij} é a inter-conectividade entre os i-ésimo e j-ésimo *clusters*, é definido

Figura 2: Exemplo de cálculo de inter-conectividade (MANCORIDIS et al., 1998).

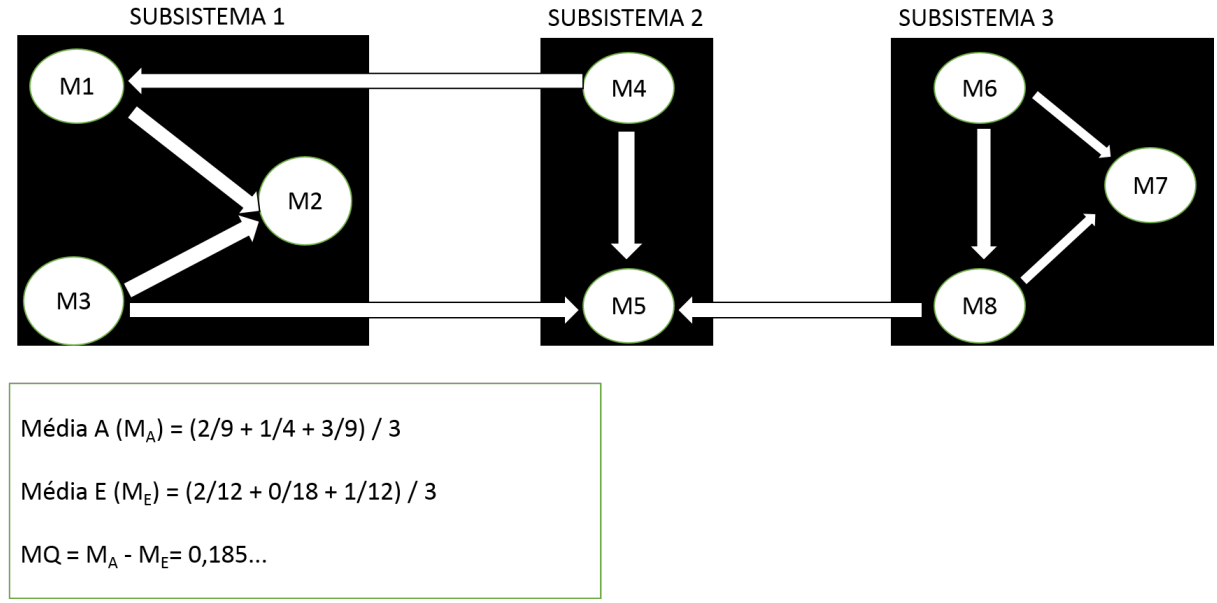


por:

$$\text{MQ Básico} = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{1}{\frac{k(k-1)}{2}}\sum_{i,j=1}^{k} E_{ij} & \text{se } k > 1 \\ A_1 & \text{se } k = 1 \end{cases} \tag{2.3}$$

Essa medida expressa o balanceamento entre inter-conectividade e intra-conectividade por meio do incentivo da criação de *clusters* coesos enquanto que a criação de *clusters* muito acoplados entre si é penalizada. A Figura 3 mostra um exemplo do cálculo do MQ. Nela existem três *clusters* identificados como subsistema 1, subsistema 2 e subsistema 3. Os valores de intra-conectividade (A) dos três *clusters* são, respectivamente, $\frac{2}{9}$, $\frac{1}{4}$ e $\frac{3}{9}$ e os valores de inter-conectividade (E) são, respectivamente, $\frac{2}{12}$, 0 e $\frac{1}{12}$.

Figura 3: Exemplo de cálculo do MQ Básico (MANCORIDIS et al., 1998).



2.1.2 Função objetivo Turbo MQ

A medida Turbo MQ possui exatamente os mesmos objetivos do MQ Básico, no entanto ela possibilita realizar o cálculo do MQ em grafos ponderados além de melhorar sua performance computacional em comparação com o MQ Básico (MITCHELL, 2003).

O MQ Turbo é calculado por meio da soma dos fatores de clusterização (em inglês, *Cluster Factor* - CF) para cada *cluster* no MDG particionado. O fator de clusterização CF_i para o *cluster* i $(1 \leq i \leq k)$, onde k é a quantidade de *clusters*, é definido como a razão entre o peso total das arestas internas e metade do peso total das arestas externas. O peso das arestas externas é dividido pela metade para penalizar igualmente ambos os *clusters* conectados pela dependência externa. Seja μ_i o número de arestas internas de um *cluster* i e $\varepsilon_{i,j}$ e $\varepsilon_{j,i}$ o número de arestas externas entre dois *clusters* distintos i e j, respectivamente. Se os pesos das dependências não forem fornecidos, todos os pesos são tratados com o valor 1 e assume-se que não existem auto-relacionamentos. As equações abaixo lembram o cálculo do MQ Turbo, apresentado na Seção 1.1. Um exemplo de cálculo do MQ Turbo é visto na Figura 4 na qual existem três *clusters* representados pelos identificadores subsistema 1, subsistema 2 e subsistema 3 que possuem respectivamente os valores de CF de $\frac{2}{3}$, $\frac{2}{5}$ e $\frac{6}{7}$.

De acordo com Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013), o fator de clusterização CF pode ser visto como uma generalização de uma medida conhecida como densidade relativa que mede a qualidade de um *cluster* em um grafo. Síma e Schaeffer (SÍMA; SCHAEFFER, 2006) provam que o problema de decisão associado ao problema de otimização que busca encontrar uma clusterização ótima com respeito à medida de densidade relativa é NP-Completo.

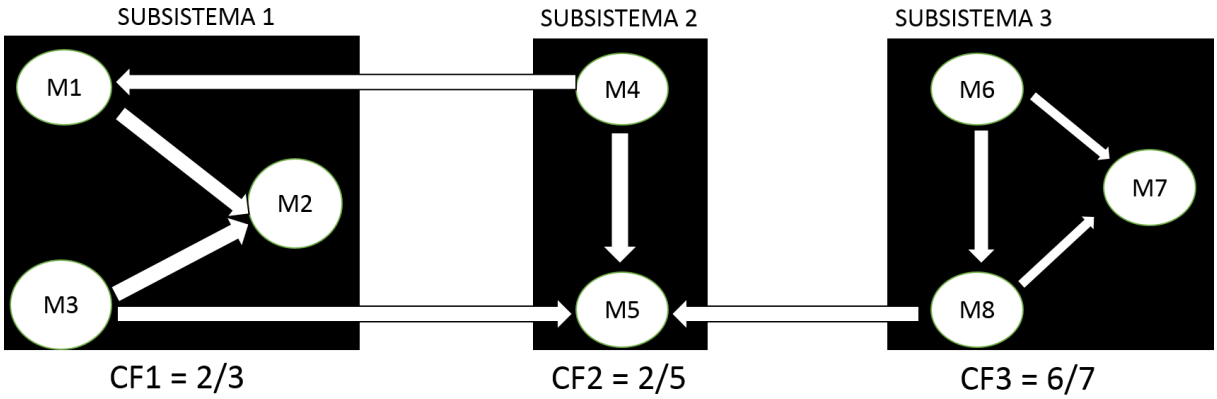
$$\text{MQ Turbo} = \sum_{i=1}^{k} CF_i \tag{2.4}$$

$$CF_i = \begin{cases} 0 & \text{se } \mu_i = 0 \\ \frac{2\mu_i}{2\mu_i + \sum_{j=1, j\neq i}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & \text{caso contrário} \end{cases} \tag{2.5}$$

2.1.3 Função objetivo Turbo MQ Incremental (ITurboMQ)

Uma otimização, também proposta por (MITCHELL, 2002), indica que um procedimento de busca que move um módulo de um *cluster* para outro não precisa recalcular os CF de todos os *clusters*. Esse movimento somente modifica os CF dos *clusters* de origem e destino envolvidos na movimentação do módulo. Sendo assim, não é necessário recalcular todos os CF quando uma movimentação dessas ocorrer: somente é necessário recalcular os CF dos *clusters* envolvidos na movimentação do módulo atualizando o valor do MQ. Essa forma de atualizar o valor de um dado MQ é chamada de *Incremental Turbo MQ*.

Figura 4: Exemplo de cálculo do Turbo MQ (MITCHELL, 2002).



2.1.4 Função objetivo Evaluation Metric (EVM)

Diferentemente de Mancoridis et al. (MANCORIDIS et al., 1998), Tucker et al. (TUCKER; SWIFT; LIU, 2001) definem uma métrica para avaliar a qualidade de uma clusterização, conhecida como *Evaluation Metric* (EVM), para cenários diferentes dos encontrados na Engenharia da Computação. O objetivo é avaliar a similaridade de dois grupos distintos de variáveis e nesse caso, grupos podem ser entendidos como *clusters* e variáveis como módulos. Apesar de compensar *clusters* com alta intra-conectividade assim com o MQ, o EVM não pune diretamente relacionamentos de módulos em diferentes *clusters*. Ao invés disso, para cada *cluster* presente, uma pontuação é definida baseada no número máximo de intra-conexões possíveis sendo que para cada intra-conexão presente, a pontuação é incrementada e, para cada intra-conexão não presente, a pontuação é decrementada.

O Algoritmo 1 detalha o pseudocódigo de cálculo do EVM conforme as ideias apresentadas por Harman (HARMAN, 2005), adaptado para receber como entrada o grafo MDG $= G(V = \{1, 2, \ldots, n\}, E)$ e uma partição $P_G = G_1, G_2, \ldots, G_k$. Considere a função $g(i)$ que, para cada vértice (módulo) $i \in V$, retorna o identificador do *cluster* onde o vértice (módulo) i se encontra.

2.2 Algoritmos genéticos de chaves aleatórias viciadas (BRKGA)

O principal objetivo desta seção é introduzir o método algoritmos genéticos de chaves aleatórias viciadas (BRKGA) (GONÇALVES; RESENDE, 2011), uma variante do método algoritmos genéticos de chaves aleatórias (RKGA) (BEAN, 1994), usado nesta dissertação para propor uma solução para o PCMS. Inicialmente, porém, introduz-se, na Seção 2.2.1, conceitos básicos da metaheurística algoritmos genéticos (AG) (GOLDBERG, 1989) e, na Seção 2.2.2, o

Algoritmo 1 Algoritmo de cálculo do EVM.

Procedimento EVM($G(V = \{1, 2, \ldots, n\}, E); P_G = G_1, G_2, \ldots, G_k$)
  Seja $EVM = 0$;
  para $u = 1$ **até** $n - 1$ **faça**
    para $v = u + 1$ **até** n **faça**
      se $g(v) = g(u)$ **então**
        se $\exists((u, v)$ **ou** $(v, u)) \in E$ **então**
          $EVM \leftarrow EVM + 1$;
        senão
          $EVM \leftarrow EVM - 1$;
  retornar EVM.

método algoritmos genéticos de chaves aleatórias (RKGA).

2.2.1 Conceitos básicos sobre algoritmos genéticos

Entre 1950 e 1960 diversos cientistas da computação estudaram, de forma independente, sistemas evolucionários com o objetivo de fazer uma analogia entre mecanismos de evolução da natureza e uma ferramenta para resolver problemas de engenharia. De todos esses estudos, conseguiu-se abstrair um conceito comum de que, para resolver um problema complexo, um conjunto de soluções candidatas deveria ser evoluído utilizando-se operadores inspirados pela seleção natural e pela variação genética. Em 1975, baseado nesses estudos, John Holland apresentou um algoritmo genético (AG) implementado como um *framework* teórico para auxiliar a resolução de problemas da Ciência da Computação (MITCHELL, 1996; REEVES, 1997). A seguir, a definição de alguns conceitos básicos comuns a diversos algoritmos genéticos e que foram utilizados na heurística definida nesta dissertação (HOLLAND, 1992; BEAN, 1994; REEVES, 1997):

1. Cromossomo: uma solução candidata de um problema específico. Normalmente o cromossomo é codificado como um vetor em $\mathbb{R}^n$, onde n é o número de elementos que compõem a solução. Por exemplo, o cromossomo $X = (4, 5, 3, 2, 1, 6, 8, 9)$ em $\mathbb{R}^8$;

2. Gene: unidade mínima de um cromossomo. No cromossomo X, apresentado acima, o quarto elemento do vetor, $X(4)$, é um gene;

3. Alelo: valor atribuído a um gene. No exemplo acima, o gene $X(4)$ possui o alelo de valor 2;

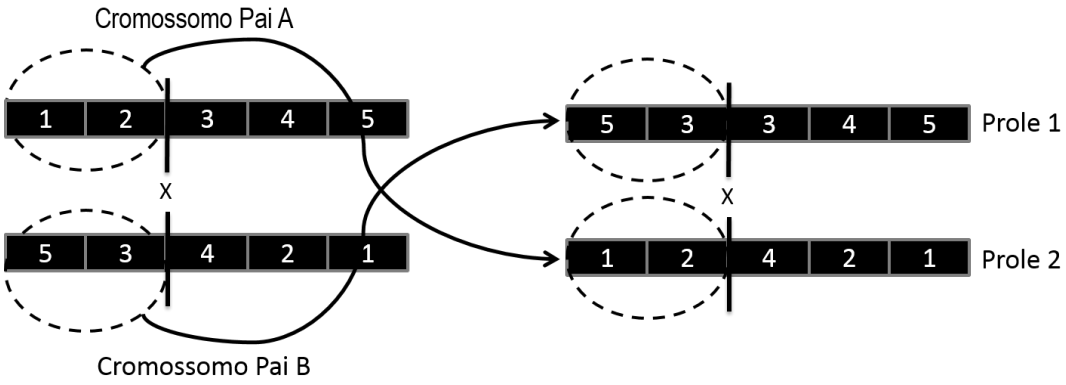
4. População: conjunto de cromossomos;

5. Prole: resultado da reprodução de 2 cromossomos;

6. Reprodução: processo de geração de uma nova população;

7. Operador de recombinação: realiza a troca de um conjunto de genes de um cromossomo por um conjunto de genes de outro cromossomo. O operador *one point crossover*, por exemplo, gera duas proles a partir de dois cromossomos-pai e um ponto de crossover $\times$ (POLI; LANGDON, 1998). A Figura 5 exemplifica esse processo para o cromossomo-pai A = (1,2,3,4,5) e para o cromossomo-pai B = (5,3,4,2,1);

Figura 5: Exemplo de *one point crossover*. Os dois cromossomos-pai estão do lado esquerdo da figura e as duas proles do lado direito. Seja $\times = 2$, a prole 1 é formada pelos alelos entre 1 e $\times$ do cromossomo-pai B concatenados com os alelos entre $\times + 1$ e n do cromossomo-pai A. A prole 2 é formada pelos alelos entre 1 e $\times$ do cromossomo-pai A concatenados com os alelos entre $\times + 1$ e n do cromossomo-pai B.



8. Operador de seleção: método de escolha de cromossomos para recombinação. O método *roullete wheel*, por exemplo, gera uma distribuição probabilística na qual a probabilidade de um cromossomo ser selecionado é proporcional à sua aptidão (KUMAR, 2012);

9. Operador de mutação: a forma mais usual de mutação escolhe aleatoriamente um subconjunto de genes, modificando seus alelos;

10. Aptidão: qualidade de um cromossomo em comparação com outros cromossomos dentro de sua população;

11. Função objetivo: função responsável por atribuir uma aptidão a um cromossomo;

12. Codificação: estratégia utilizada para construir um cromossomo a partir de informações específicas do problema a ser tratado; e

13. Decodificação: estratégia utilizada para criar um algoritmo determinístico que recebe como entrada um cromossomo e associa a ele uma solução para o problema em questão.

O Algoritmo 2 organiza os conceitos relacionados acima em um *framework* teórico genérico, assim como proposto por Holland (HOLLAND, 1992), no qual a maioria dos algoritmos baseados em AG da literatura (inclusive os utilizados para resolver o PCMS) instanciaram para propor métodos de resolução para seus problemas específicos (REEVES, 1997).

Algoritmo 2 Algoritmo genético genérico

```
Procedimento AG GENÉRICO
    Escolha uma população inicial de cromossomos;
    enquanto critério de parada não tiver sido satisfeito faça
        enquanto não existirem proles suficientes faça
            Selecionar os pais para reprodução;
            Escolher os parâmetros de recombinação;
            Realizar a recombinação;
            Realizar a mutação;
            Analisar a aptidão das proles;
        Selecionar nova população;
    retornar A melhor solução encontrada.
```

Em relação aos conceitos apresentados acima cabe ressaltar que o tamanho de uma população afeta a eficácia e o desempenho de um algoritmo genético (REEVES, 1997; FOGEL, 1995; WHITLEY, 1994). Se, por um lado, a aptidão das soluções tende a ser muito baixa se a população for muito pequena, por outro, o desempenho do algoritmo tende a ser ruim caso a população seja muito grande. Não existe um consenso na literatura a respeito do tamanho de uma população ideal. No entanto, estudos empíricos mostram que uma população de tamanho entre o intervalo de 10 a 160 tem alta probabilidade de encontrar resultados satisfatórios (REEVES, 1997; FOGEL, 1995). Em relação à estratégia de reprodução, o *framework* teórico de (HOLLAND, 1992) indica que a cada geração toda a população deve ser substituída por novos cromossomos. Do ponto de vista de otimização e de evolução, essa estratégia não é interessante pois o custo computacional de ter gerado soluções com alta aptidão é todo desperdiçado: "bons genes" deixam de ser propagados ao longo das gerações. Para contornar essa situação, Jong (JONG, 1975) introduz o conceito de elitismo que garante a sobrevivência de indivíduos mais aptos ao longo das gerações. No entanto, a estratégia de substituição total pode ser utilizada como um bom critério de perturbação, em que após uma quantidade pré-definida de gerações sem melhoria em relação aos cromossomos com maior aptidão, uma reinicialização desse tipo possa ser utilizado, conforme pode ser observado em Morán-Mirabal et al. (MORÁN-MIRABAL et al., 2013). Por fim, muito discute-se em relação a taxas de mutação sem um consenso a respeito de quantos mutantes devem ser introduzidos em uma nova população (REEVES, 1997). Segundo Harvey (HARVEY, 1992), uma boa probabilidade de mutação (p_m), aplicada a diversos problemas com relativo sucesso, é $p_m = 0{,}01$ ou $p_m = 1/l$, sendo l o tamanho de um cromossomo.

2.2.2 O *framework* algoritmos genéticos de chaves aleatórias (RKGA)

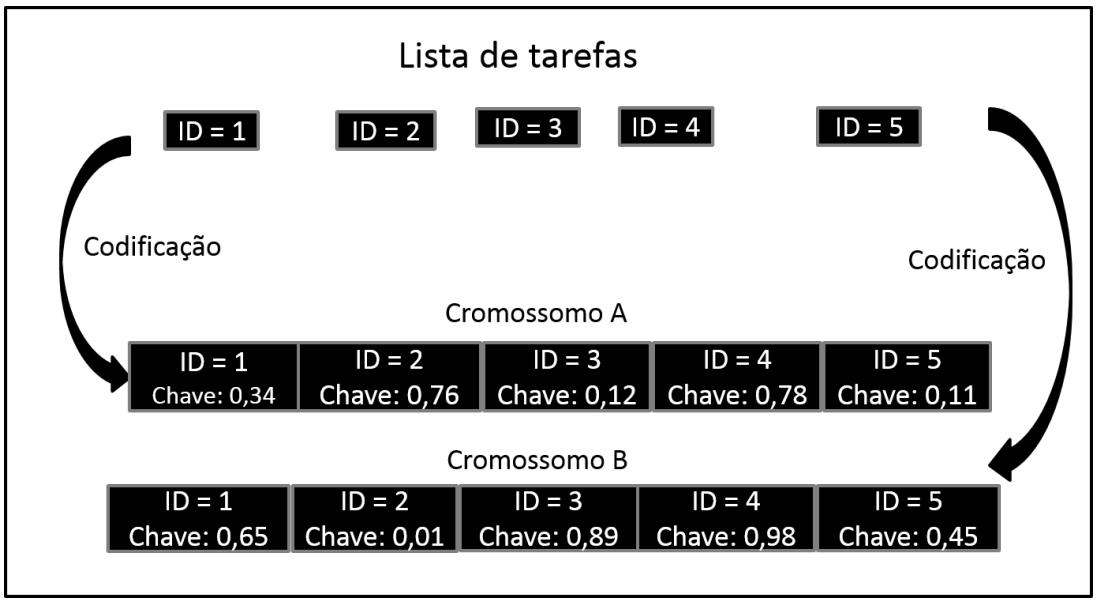
Conforme explicado na seção anterior, o operador de recombinação realiza a troca de genes entre dois cromossomos. O operador *one point crossover* é um dos operadores mais tradicionais (BEAN, 1994). No entanto, constantemente as proles geradas por esse operador não são soluções viáveis para um determinado problema. Analisando um simples problema de sequenciamento onde a permutação de um conjunto de valores é uma solução viável, o

problema da viabilidade torna-se bastante claro. Utilizando o mesmo exemplo da Figura 5 é fácil perceber que a prole $2 = (1, 2, 4, 2, 1)$ não é uma solução viável já que dois de seus genes estão repetidos, inviabilizando a sequência obtida como uma permutação, que é uma solução válida. De forma a evitar esse problema, diversos autores criaram representações fortemente acopladas aos problemas que visavam resolver, como pode ser visto em (GOLDBERG; LINGLE, 1985; GREFENSTETTE et al., 1985; GREFENSTETTE, 1987; CLEVELAND; SMITH, 1989). Em especial, Bean (BEAN, 1994) propôs uma estratégia genérica para resolver tal problema, apresentada na próxima seção.

Para resolver o problema da viabilidade, Bean (BEAN, 1994) utiliza chaves aleatórias geradas no intervalo de $[0, 1]^n$, onde n representa o número de genes em um cromossomo.

De forma a ilustrar a robustez do mecanismo de chaves aleatórias, utiliza-se um exemplo de um simples problema de programação de tarefas em uma máquina para o qual, dado um conjunto $N = \{1, 2, \ldots, n\}$ de n tarefas, uma sequência de tarefas é uma solução viável. Uma sequência de tarefas pode ser representada por uma permutação. Considere uma instância desse problema com $n = 5$ tarefas, $N = \{1, 2, 3, 4, 5\}$. O método de chaves aleatórias cria dois cromossomos compostos por cinco genes onde cada alelo possui uma chave aleatória gerada no intervalo de $[0, 1]^n$, que representa as tarefas mencionadas por meio de seus índices. Essa transformação de variáveis inerentes do problema em números aleatórios é denominada de codificação. A Figura 6 ilustra essa codificação.

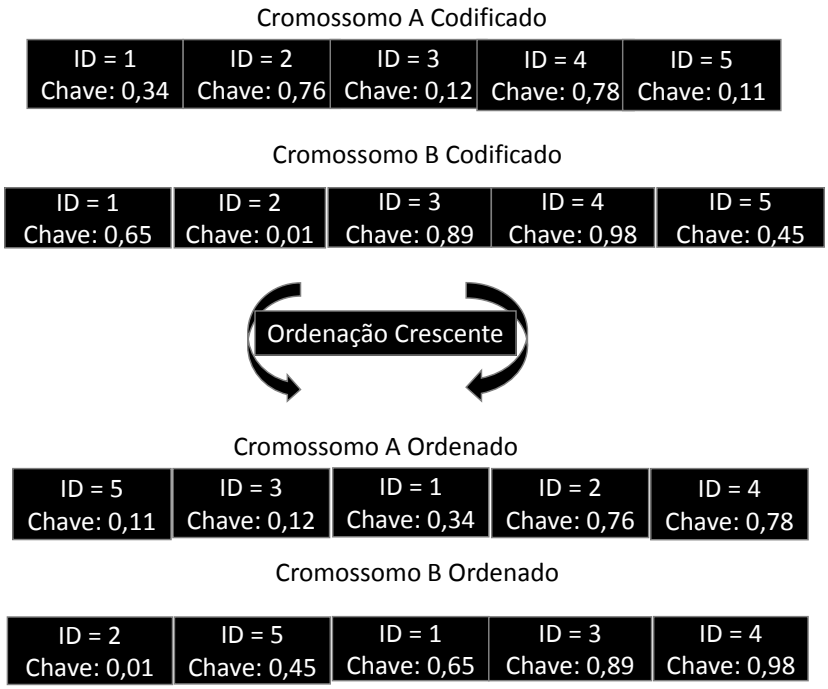
Figura 6: Codificação utilizando chaves aleatórias. Nos dois cromossomos abaixo cada tarefa possui associada uma chave aleatória: a tarefa de ID 4 no cromossomo A possui chave de valor 0,78 e no cromossomo B possui chave de valor 0,98.



Para obter as soluções codificadas pelos cromossomos A e B da Figura 6 basta ordenar crescentemente as chaves aleatórias e a permutação dos IDs das tarefas representarão a solução para o problema, conforme pode ser visto na Figura 7. Ou seja, para o problema de sequencia-

mento o exemplo possui duas soluções ou dois sequenciamentos de tarefas representados pelos IDs do “Cromossomo A Ordenado” e do “Cromossomo B Ordenado”, a saber: as permutações (5,3,1,2,4) e (2,5,1,3,4).

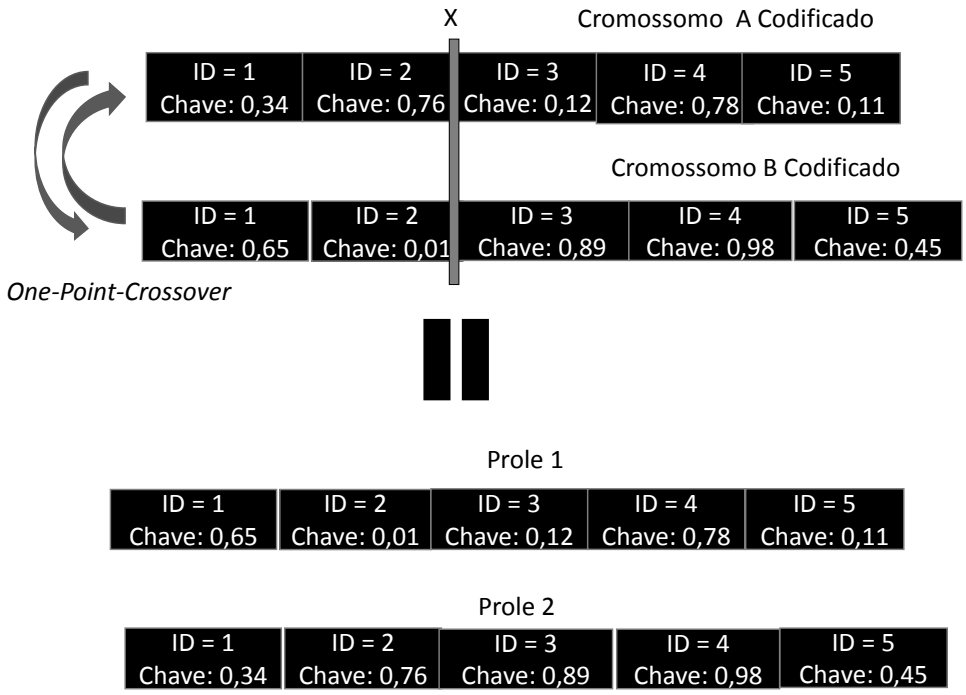
Figura 7: Decodificação por meio da ordenação das chaves aleatórias. Os cromossomos são ordenados crescentemente a partir de suas chaves aleatórias, gerando uma permutação de seus “IDs”.



Ao se aplicar o *one point crossover* nos cromossomos A e B, com $\times = 2$, são obtidas duas proles cujos genes são prole $1 = (0{,}65; 0{,}01; 0{,}12; 0{,}78; 0{,}11)$ e prole $2 = (0{,}34; 0{,}76; 0{,}89; 0{,}98; 0{,}45)$, conforme ilustrado na Figura 8. Percebe-se, que as proles 1 e 2 não herdam nenhuma característica do problema a ser tratado (os identificadores foram todos reconfigurados), ou seja, a estratégia de recombinação atua exclusivamente sobre as chaves aleatórias não levando em conta nenhuma especificidade do problema em questão, a qual foi totalmente delegada para o processo de decodificação. Nesse ponto reside toda a robustez da estratégia, fazendo com que a mesma possa ser aplicada em qualquer problema combinatório, sendo necessário apenas definir as estratégias de codificação e decodificação, essas sim, específicas do problema a ser tratado.

Por fim, ao se ordenar crescentemente as chaves aleatórias das proles 1 e 2 tem-se uma solução viável para o problema em questão, eliminando totalmente o problema de viabilidade apresentado na seção anterior, já que o resultado obtido por essas soluções será uma permutação dos índices representados pelas chaves aleatórias, como pode ser visto na Figura 9. Resumindo, se qualquer vetor de chaves aleatórias pode derivar uma solução viável, então qualquer prole gerada

Figura 8: *One point crossover* em cromossomos de chaves aleatórias.



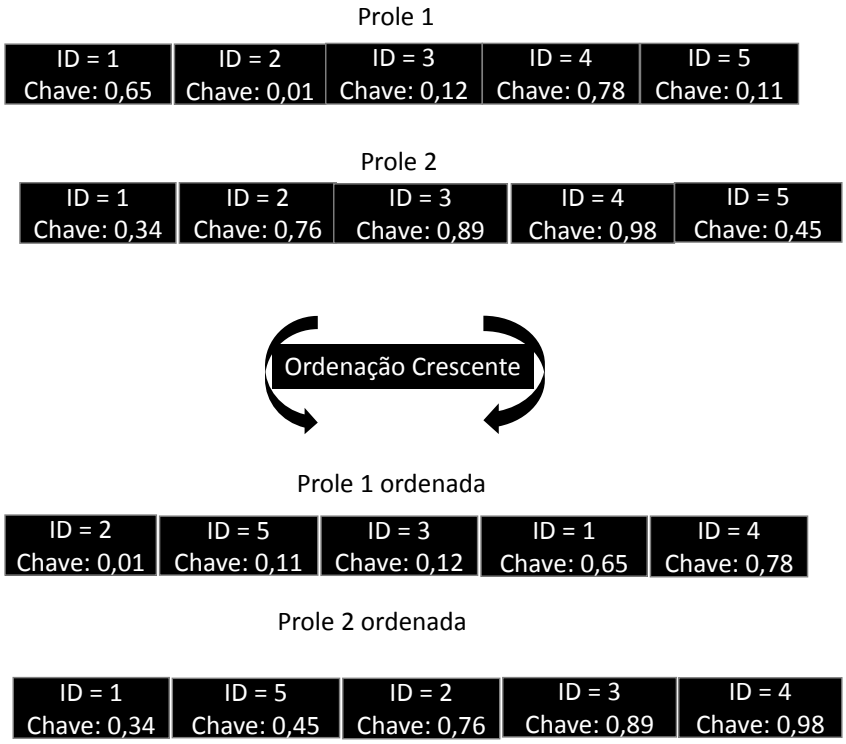
por técnicas de recombinação também será uma solução viável (GONÇALVES; RESENDE, 2011).

2.2.2.1 Seleção, recombinação e mutação

Segundo Gonçalves e Resende (GONÇALVES; RESENDE, 2014), o RKGA evolui uma população de cromossomos por meio de um número de iterações chamado de "gerações". Sua população inicial é formada de p cromossomos onde cada alelo é gerado de forma aleatória no intervalo real de [0,1]. Depois que o decodificador transforma os números aleatórios em soluções para o problema em questão, a aptidão de cada cromossomo é calculada por sua função objetivo. Por último, a população é particionada em dois grupos: um pequeno grupo elite p_e de cromossomos (aqueles que possuem melhor aptidão) e um conjunto de cromossomos não elite $p - p_e$ onde $p > 2 \times p_e$.

A partir da população inicial, uma nova geração de cromossomos deve ser criada e, para isso, o RKGA utiliza uma estratégia elitista na qual todos os cromossomos elite da geração k são copiados para a geração $k + 1$. Para continuar compondo a nova geração, uma quantidade p_m de mutantes é criada. Esses mutantes são vetores de chaves aleatórias gerados da mesma forma que a população inicial (cada alelo do cromossomo mutante é gerado de forma aleatória no intervalo real de [0,1]). Finalmente, na geração $k + 1$ com p_e cromossomos elite e p_m mutantes, é necessário adicionar p - (p_e + p_m) cromossomos para compor toda a população. Esses últimos

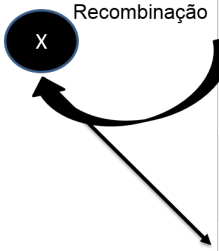
Figura 9: Proles decodificadas por meio da ordenação crescente. As proles são ordenadas crescentemente a partir de suas chaves aleatórias, gerando uma permutação de seus "IDs".



cromossomos são gerados por meio de recombinação dos cromossomos já existentes usando a técnica de *parametrized uniform crossover* (SPEARS; JONG, 1991) ao invés de operadores mais tradicionais como *one point crossover*. Nessa técnica, após dois cromossomos pais serem escolhidos aleatoriamente de toda a população, um número aleatório ("dado aleatório viciado") é gerado no intervalo de [0,1]. Caso esse número seja maior do que o valor do parâmetro "vício probabilístico" (v_p), por exemplo 70%, o alelo do primeiro cromossomo-pai deve ser escolhido. No entanto, se o valor for menor do que 70%, o alelo do segundo cromossomo-pai deve ser escolhido. A Figura 10 exemplifica esse método de recombinação.

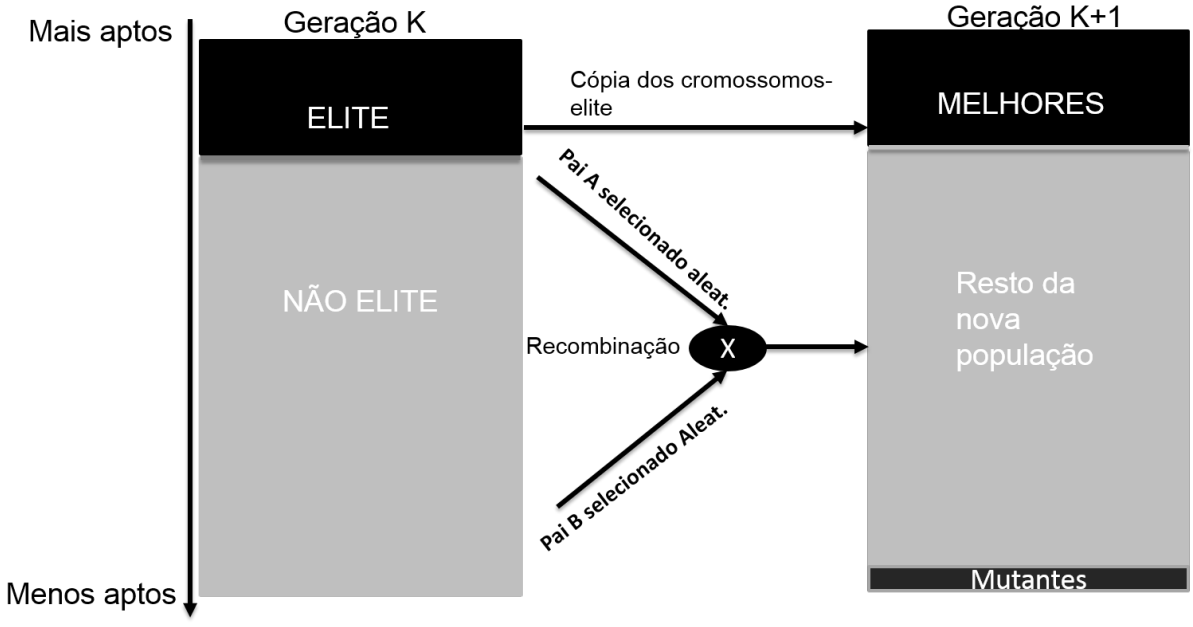
A Figura 11 ilustra essa dinâmica da evolução: a esquerda, situa-se a população atual. Depois que todos os indivíduos foram ordenados de acordo com sua aptidão, um subcojunto elite é formado (identificado pelo rótulo ELITE) contendo p_e e o resto da população forma o subconjunto de cromossomos não elite (identificado pelo rótulo NAO ELITE). O subconjunto elite é copiado para a próxima população em sua íntegra, sendo identificado pelo rótulo MELHORES. De forma a evitar uma rápida convergência para ótimos locais, uma quantidade de mutantes é inserida (rótulo Mutantes) e o resto da nova população é formada pelo processo de recombinação *parametrized uniform crossover*.

Figura 10: *Parametrized Uniform Crossover.* Nas primeiras duas linhas da tabela tem-se os dois cromossomos-pai a serem recombinados. Na terceira linha, o valor do dado aleatório viciado que irá decidir de qual pai a prole herdará o gene. Na quarta linha, a indicação da relação entre o dado aleatório e o vício probabilístico de herança e, na última linha, a prole. Na coluna 4 o dado aleatório viciado possui um valor de $0{,}87$, ou seja, maior do que o vício probabilístico de $0{,}7$ fazendo com que a prole herde o gene do cromossomo-pai B. Se o valor do dado fosse menor do que $0{,}7$, o gene seria herdado do cromossomo-pai A.



Cromossomo 1	0.46	0.91	0.33	0.75	0.51
Cromossomo 2	0.84	0.32	0.64	0.04	0.48
Número Aleatório	0.34	0.69	0.87	0.65	0.99
Vício de 0.7	<	<	>	<	>
Prole	0.46	0.91	0.64	0.75	0.48

Figura 11: RKGA para problemas combinatórios.



2.2.3 O *framework* algoritmos genéticos de chaves aleatórias viciadas (BRKGA)

O BRKGA é um *framework* genérico de metaheurísticas utilizado para construção de heurísticas para resolver problemas de otimização combinatória baseado no RKGA apresentado na seção anterior.

O BRKGA funciona exatamente igual ao RKGA no que diz respeito a recombinação

e a mutação. A principal diferença entre ambos está na seleção dos pais para a recombinação. O BRKGA seleciona aleatoriamente um cromossomo-pai do subconjunto ELITE e um do subconjunto NAO ELITE, enquanto o RKGA seleciona os pais de toda a população (Figura 11). Essa pequena diferença faz com que um indivíduo com uma alta aptidão tenha uma maior probabilidade de passar adiante seus "bons genes" para as futuras proles. Contribuindo com esse mecanismo elitista está a estratégia de recombinação utilizada no BRKGA que, assim como a estratégia utilizada por Bean (BEAN, 1994), utiliza o *parametrized uniform crossover* (JONG, 1975) já explicado na seção anterior. Nesse caso, o fato de v_p ser um valor que assume valores maiores que $0{,}5$ garante que a prole gerada por esse método tenha uma probabilidade maior de herdar um gene do seu cromossomo pai mais apto, pois no BRKGA o pai proveniente do grupo ELITE é sempre aquele influenciado pela maior probabilidade definida por v_p (no exemplo da Figura 10 o cromossomo A seria proveniente do grupo ELITE pois a probabilidade desse de propagar seus genes é de 70%, enquanto que a do cromossomo B é de 30%).

No BRKGA existe uma clara divisão entre a parte genérica da solução, independentemente de qualquer idiossincrasia do problema de natureza combinatória, e a parte específica de cada problema. Como pode ser visto na Figura 12, a parte independente não possui nenhum conhecimento da parte dependente do problema. A única interface entre ambas é o processo de decodificação que produz soluções a partir dos vetores de chaves aleatórias e calcula o valor das respectivas aptidões. Sendo assim, para especificar uma heurística baseada no BRKGA basta apenas definir a representação dos cromossomos (codificação) e a transformação desses cromossomos em possíveis soluções para o problema combinatório (decodificação), além de escolher valores para os parâmetros específicos do BRKGA, como o tamanho da população e o número de gerações, por exemplo.

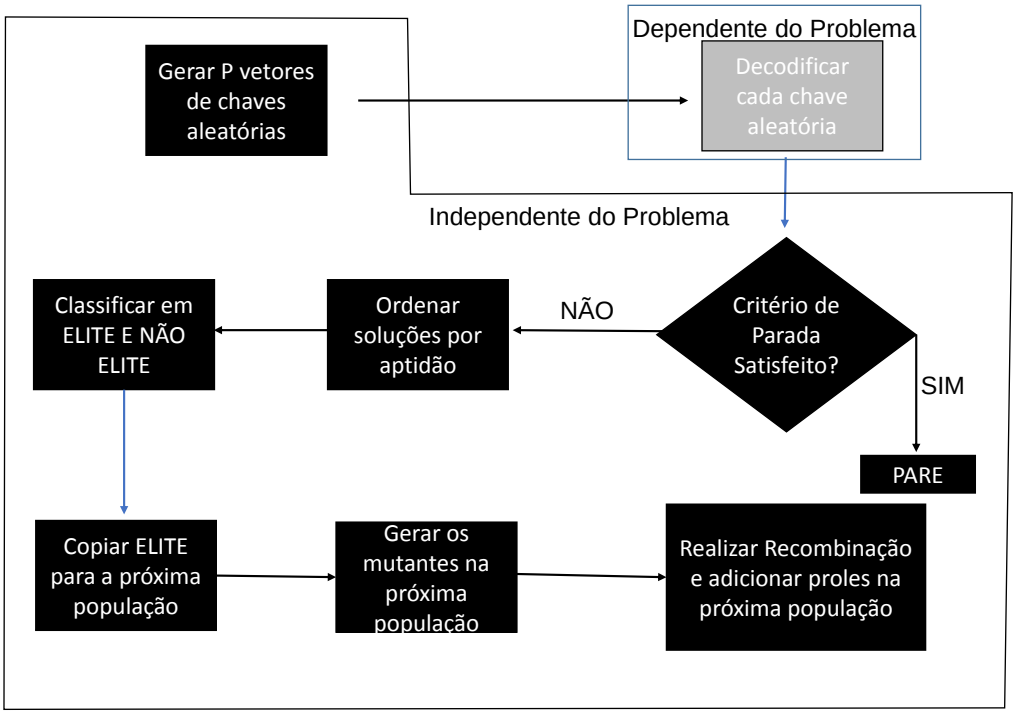
Essa estratégia permite um alto reuso de software já que a parte genérica pode ser reaproveitada para a resolução de qualquer problema combinatório, permitindo que os projetistas de algoritmos possam focar na construção das partes específicas do problema a ser resolvido.

2.2.3.1 Eficiência do BRKGA

Nesta seção, apresentam-se os resultados de alguns experimentos conduzidos por Gonçalves e Resende (GONÇALVES; RESENDE, 2014) com o objetivo de mostrar a eficiência do BRKGA. Os experimentos utilizam a ferramenta de plotagem *time to target* (TTTPLOTS) (AIEX; RESENDE; RIBEIRO, 2007) que caracteriza o tempo de execução de algoritmos estocásticos utilizados em problemas de otimização combinatória. Esse tipo de plotagem vem sendo largamente utilizada por Feo et al. (FEO; RESENDE; SMITH, 1994). No gráfico com as plotagens *time to target*, o eixo vertical mostra a probabilidade que um algoritmo tem de encontrar uma solução tão boa quanto uma solução alvo pré-definida dentro de um intervalo de tempo especificado, esse representado no eixo horizontal.

Experimento conduzido por Gonçalves e Resende (GONÇALVES; RESENDE, 2014)

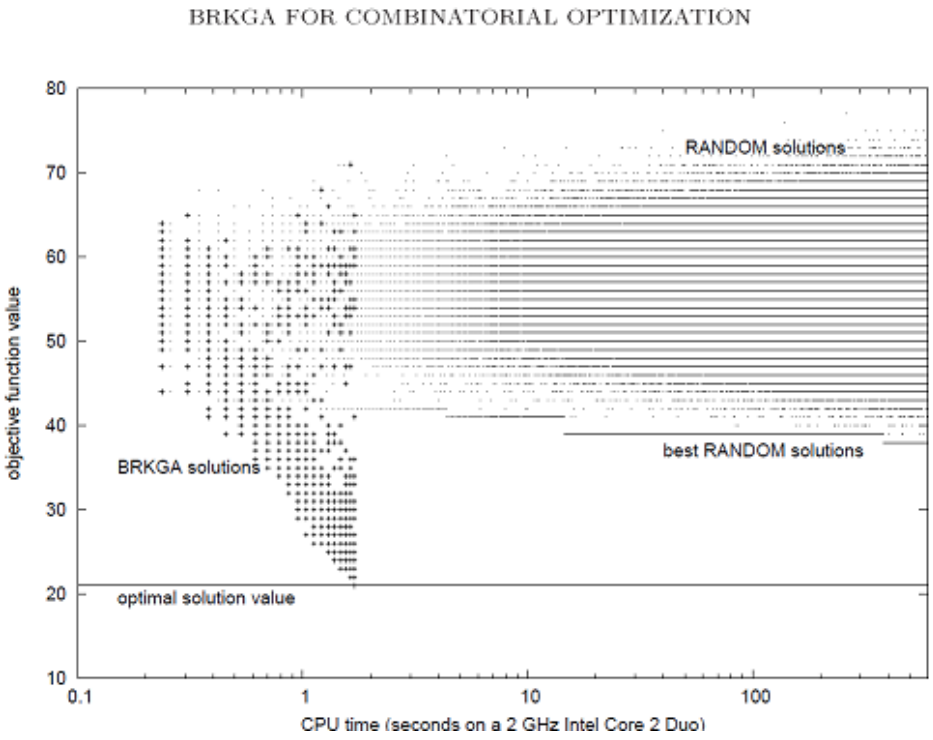
Figura 12: BRKGA para problemas combinatórios. Nesta figura os retângulos em azul representam as partes fixas do *framework* BRKGA, ou seja, a parte genérica independente do problema a ser resolvido, e o retângulo vermelho representa a parte específica que irá variar dependendo do problema combinatório a ser resolvido.



compara a execução de um BRKGA para o problema de *covering-by-pairs* (BRESLAU et al., 2011) com uma heurística puramente randômica. O resultado da comparação é mostrado na Figura 13. É possível notar que em menos de dois segundos o BRKGA consegue atingir a solução ótima enquanto que a melhor solução puramente randômica ainda está muito distante da solução ótima, mesmo após 600 segundos de execução. Esse experimento permite concluir que, apesar de utilizar apenas chaves aleatórias, o BRKGA é muito mais eficiente em encontrar soluções ótimas ou quase ótimas do que algoritmos puramente randômicos.

Conforme mencionado na seção anterior, um BRKGA seleciona um cromossomo-pai para recombinação do subconjunto de cromossomos elite e um pai do resto dos cromossomos não elite, diferentemente do RKGA que seleciona randomicamente os pais de toda a população. O impacto dessa pequena modificação influencia bastante a qualidade da solução gerada por ambos os métodos. Os algoritmos baseados no BRKGA possuem a tendência de encontrar soluções melhores do que os baseados no RKGA, dado um mesmo tempo de execução. Utilizando novamente o exemplo do problema *covering-by-pairs*, outro experimento foi conduzido por Gonçalves e Resende (GONÇALVES; RESENDE, 2011) realizando a comparação do tempo para obtenção da solução ótima de três heurísticas, duas delas baseadas no RKGA e uma baseada no BRKGA. De acordo com a Figura 14 é perceptível a eficiência do BRKGA quando comparado com os dois outros métodos. Por exemplo, no segundo 325, o tempo que o RKGA

Figura 13: BRKGA comparado com soluções puramente randômicas. No gráfico, os valores do BRKGA estão representados como uma "cruz" (+). É perceptível que em 2 segundos o BRKGA atinge o resultado alvo definido (*optimal solution value*) enquanto que as soluções randômicas ainda estão longe deste valor alvo (GONÇALVES; RESENDE, 2014).



.

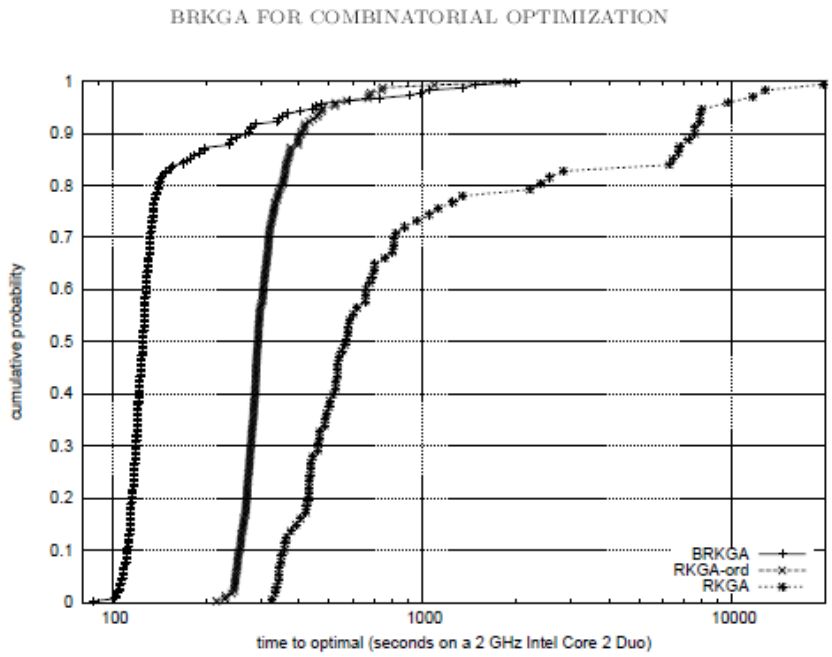
leva para resolver uma de suas 84 tentativas, o BRKGA resolve 184 de suas 200 tentativas. Outras comparações mostrando a superioridade do BRKGA podem ser visualizadas em Gonçalves et al. (GONÇALVES; RESENDE; TOSO, 2014).

2.3 Trabalhos que abordam o PCMS

2.3.1 Algoritmos exatos

O primeiro método a resolver de forma ótima o PCMS foi proposto por Mancoridis et al. (MANCORIDIS et al., 1998) no qual todas as possibilidades de clusterização de um MDG eram obtidas e comparadas até que se obtivesse a clusterização com o maior MQ (método exaustivo de busca). A técnica exata de Mancoridis et al. (MANCORIDIS et al., 1998) somente conseguiu resolver sistemas com até 15 módulos já que o número de clusterizações possíveis de se encontrar a partir de um MDG cresce exponencialmente em relação ao número de módulos. Para se calcular o total de clusterizações possíveis em um MDG considere n o número de módulos no MDG e k o número de *clusters* em uma dada clusterização. Tem-se que $1 \leq k \leq n$. Seja $S(n, k)$ o número total de k-$parties$ distintas para um MDG com n elementos, isto é, o número de partições (soluções) possíveis com n módulos e k *clusters*. O número total de

Figura 14: Heurística baseada no BRKGA comparada com duas heurísticas baseadas no RKGA (GONÇALVES; RESENDE, 2011).



k-partições distintas satisfaz a equação:

$$S(n,k) = \begin{cases} 1 & \text{se } k = 1 \text{ ou } k = n \\ S_{n-1,k-1} + kS_{n-1,k} & \text{caso contrário} \end{cases} \tag{2.6}$$

Para se conhecer o número de soluções distintas do PCMS em um MDG com n módulos é necessário calcular a soma de $S(n,k)$ para $k = 1, \ldots, n$, dada por $\sum_{k=1}^{n} S(n,k)$. Por exemplo, um MDG de cinco módulos possui $S(5,1) = 1$ solução com um *cluster* mais $S(5,2) = 15$ soluções distintas com dois *clusters* mais $S(5,3) = 25$ soluções distintas com três *clusters*, e assim sucessivamente, totalizando 52 soluções distintas. O mesmo cálculo pode ser feito para um MDG com 15 módulos que possui 1.382.958.545 soluções distintas, ou seja, o crescimento de *clusters* é limitado por $O(n!)$ sendo n o número de módulos de um software (MANCORIDIS et al., 1998). Isto significa que sistemas maiores apresentaram um espaço de busca muito grande que não pôde ser explorado em um tempo razoável pelo método exaustivo implementado por Mancoridis et al. (MANCORIDIS et al., 1998).

Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013) utilizam três técnicas de formulação matemática para resolver o PCMS: a primeira formula-o como a soma de funções fracionais lineares e as outras duas utilizam a técnica de programação linear inteira mista. Com essas formulações os autores conseguem resolver de forma ótima 45 instâncias incluindo todas as 8 instâncias resolvidas de forma ótima por (MANCORIDIS et al., 1998).

Mais recentemente, Kramer et al. (KRAMER et al., 2014) apresentam outra técnica de formulação matemática baseada na abordagem *branch-and-price* (VANDERBECK, 2011)

também com o intuito de resolver de forma ótima o PCSM. Essa técnica é aplicada em 45 instâncias da literatura sendo que 15 delas foram propostas por Mitchell et al. (MITCHELL, 2002) e as 30 restantes foram propostas por Mamaghani et al. (MAMAGHANI; MEYBODI, 2009). O resultado apresentado pelos autores mostra que 43 instâncias foram resolvidas de forma ótima (11 instâncias a mais do que os resultados apresentados por Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013)). Em termos de eficiência, os resultados quando comparados com os resultados de Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013) mostram uma melhora no tempo de execução de 99,1% para instâncias grandes, 99,6% para instâncias médias e 92% para instâncias pequenas.

2.3.2 Algoritmos genéticos não interativos

Praditwong (PRADITWONG, 2011) realiza um estudo experimental utilizando dois tipos de algoritmos genéticos para resolver o PCMS, cada um deles com uma diferente representação de cromossomo. O primeiro algoritmo usa uma representação de cromossomo denominada *Group Number Encoding* (GNE), a qual define, para cada módulo, o identificador de um *cluster*. Por exemplo, a solução candidata $(4\,2\,4\,2\,1\,3\,3)$ indica que o primeiro e o terceiro módulos pertencem ao *cluster* de identificador 4, o segundo e o quarto módulos pertencem ao *cluster* 2 e assim por diante. É fácil perceber que essa estratégia de representação é extremamente redundante pois o sequencial $(1\,3\,1\,3\,2\,4\,4)$ representa a mesma solução que o sequencial $(4\,2\,4\,2\,1\,3\,3)$. O segundo algoritmo genético é baseado nas técnicas de agrupamento definidas por Falkenauer (FALKENAUER, 1996) e denominada *Grouping Genetic Algorithms* (GGA). Essa técnica utiliza dois componentes para representar uma solução para o PCMS. O primeiro mapeia cada módulo para um *cluster* enquanto que o segundo componente define os *clusters* existentes. Por exemplo, se existir quatro possíveis *clusters* A,B,C,D e se existir o mapeamento A $= 1, 3$, B $= 2, 4$, C $= 5$ e D $= 6, 7$ o mapeamento A indica que os módulos 1 e 3 pertencem ao *cluster* A, o mapeamento B indica que os *clusters* 2 e 4 pertencem ao *cluster* B e assim por diante. Essas duas técnicas foram aplicadas em 17 instâncias da literatura e os resultados mostram que para instâncias ponderadas e não ponderadas há uma superioridade da técnica GGA em relação à qualidade das soluções obtidas e em relação à sua performance.

O grupo da Drexel University desenvolveu uma ferramenta denominada Bunch (MANCORIDIS et al., 1999) na qual é permitido selecionar alguns métodos para resolver o problema do PCMS, entre eles um algoritmo genético. Esse algoritmo genético utiliza o método de codificação GNE com uma taxa de mutação de $0.004 \times log_2 n$, uma taxa de recombinação de 80% para populações de 100 indivíduos ou menos, e uma taxa de 100% de recombinação para populações de 1.000 indivíduos ou mais. O tamanho da população utilizada é $10 \times n$ sendo n o número de módulos do software a ser clusterizado e o número de gerações é de $200 \times n$.

As primeiras versões da ferramenta Bunch possuem três principais limitações identificadas por desenvolvedores entrevistados pelo grupo da Drexel University: alguns módulos

em um software acabam possuindo referências a diversos outros módulos em diversos *clusters*, ou seja, acabam sendo transversais a todo o sistema gerando uma dificuldade muito grande de atingirem os objetivos de mínimo acoplamento e máxima coesão por meio da clusterização. A estratégia para mitigar esse problema é identificar esses módulos transversais e os colocar em um único *cluster*. A outra limitação identificada é a geração de *clusters* geradores de conflitos com a clusterização realizada por um arquiteto de software conhecedor do sistema em questão. Obviamente, os arquitetos podem estar errados, no entanto, segundo Mancoridis et al. (MANCORIDIS et al., 1999), é mais provável que a ferramenta Bunch esteja gerando os conflitos com os resultados oriundos da análise subjetiva dos arquitetos por dois motivos: a ferramenta Bunch gera resultados sub-ótimos e o fato do MQ só levar em conta a topologia do grafo que representa o software e não os aspectos de uma clusterização baseada em semântica. Para solucionar essas duas limitações as versões posteriores do Bunch passaram a permitir a modificação manual de alguns módulos de acordo com o conhecimento tácito de arquitetos dos sistemas a serem clusterizados. Além de permitir a influência manual no processo automatizado, tal modificação diminui o espaço de busca das heurísticas pois um desenvolvedor poderia criar uma regra na qual dois módulos específicos poderiam estar sempre no mesmo *cluster*. A última limitação diz respeito a uma mudança muito drástica em softwares já clusterizados quando pequenas evoluções são realizadas. A solução para essa última limitação é permitir uma clusterização incremental na qual alguns *clusters* ficariam "congelados" enquanto outros *clusters* poderiam ser modificados.

Dias e Ochi (DIAS; OCHI, 2003) realizam um estudo em instâncias geradas artificialmente buscando os limites de melhoria da eficácia de algoritmos genéticos utilizados para tratar problemas de clusterização sem aumentar consideravelmente seu tempo de execução. Os autores apresentam sete versões do algoritmo genético proposto por Mancoridis et al. (DOVAL; MANCORIDIS; MITCHELL, 1999) que atua em grafos não ponderados utilizando o MQ para avaliar a qualidade das soluções geradas. A seguir, um breve resumo das principais melhorias propostas por Dias e Ochi (DIAS; OCHI, 2003):

- Busca local: aplicação do procedimento de busca local no melhor indivíduo de uma geração logo após a aplicação do operador de mutação. O objetivo dessa busca é refinar a clusterização realizada por meio da troca de módulos entre os *clusters*. Foi verificado que ao aplicar o procedimento em todos os indivíduos da geração o custo computacional torna-se muito alto já que o MQ teria que ser recalculado diversas vezes e, por isso, o procedimento só é aplicado no melhor indivíduo de uma geração. De forma resumida, para cada um dos módulos, o procedimento de busca local identifica o *cluster* no qual o módulo compartilha o maior número de dependências. Se esse *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 da aptidão a troca é realizada;

- Calibramento dos *clusters*: considerando nc o número de *clusters* da melhor solução obtida até a geração corrente, e q um número aleatório que irá definir o número de *clusters* a

serem modificados pelo operador de mutação, Dias e Ochi (DIAS; OCHI, 2003) definem a fórmula: $1 \leq q \leq (1.1 \times nc)$. O objetivo desse calibramento é progressivamente ajustar o número máximo de *clusters* alterados pelo operador genético;

- Busca local com reaplicação: a ideia dessa melhoria é a reaplicação da busca local logo após uma primeira aplicação da busca e consequente reconfiguração conduzida por ela. A ideia é tentar gerar alguma nova melhoria de qualidade do MQ a partir da nova configuração realizada pela primeira aplicação do procedimento de busca local; e

- Diversificação: procedimento de diversificação das soluções, onde a melhor solução da geração corrente é replicada m vezes, sendo m o tamanho da população. Nessas m replicações, são definidos aleatoriamente: o número de genes a serem modificados, suas posições no cromossomo e seus novos valores. O objetivo dessa estratégia é investigar o espaço de busca perto de soluções consideradas mais aptas. Esse mecanismo é acionado sempre que ocorre uma melhoria na melhor solução obtida pelo genético e na geração da população inicial.

Os resultados de Dias e Ochi (DIAS; OCHI, 2003) mostram que todas as versões criadas com as melhorias detalhadas acima superaram o algoritmo genético básico definido por Mancoridis et al. (DOVAL; MANCORIDIS; MITCHELL, 1999) chegando a aproximadamente 3% da aptidão dos melhores resultados conhecidos da literatura.

Botelho et al. (BOTELHO; SEMAAN; OCHI, 2011) também realizam diversos experimentos com instâncias reais e artificiais com algoritmos genéticos buscando encontrar soluções com maior qualidade do que as soluções obtidas por algoritmos genéticos tradicionais. Seus experimentos variam a forma de construir as soluções iniciais utilizando um método totalmente aleatório e um método baseado no peso das arestas do grafo, que representa a solução em questão, e variam a utilização de uma busca local e de uma estratégia de reconexão de caminhos (GLOVER; KOCHENBERGER, 2003). Em relação aos operadores genéticos, o método de codificação utilizado é o *Group Number Encoding*, a seleção utilizada foi a seleção por torneio de duas soluções (HOLLAND, 1992), e o cruzamento utilizou o *one point crossover*. Seus resultados mostram que a utilização da busca local obteve resultados satisfatórios de MQ em um tempo computacional relativamente baixo. Alternativamente, a estratégia de reconexão de caminhos obteve um alto tempo computacional e valores de MQ menores do que os obtidos pelas demais estratégias.

Pinto (PINTO, 2014) propõe uma solução para o PCMS utilizando a metaheurística *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003) e realizam experimentos com variantes de algoritmos genéticos e buscas locais. Os experimentos foram realizados utilizando 46 instâncias de softwares *open source* ou *free softwares*. Em seus experimentos com algoritmos genéticos, foram avaliadas 20 configurações sendo 14 utilizando a estratégia de codificação *Group Number Encoding* (GNE) e outras seis utilizando a estratégia de codificação

Grouping Genetic Algorithm (GGA). As configurações para a codificação GNE variam o método de recombinação (*one point crossover* e *two point crossover*) e o método de seleção (torneio ou classes). As configurações para o GGA variam a estratégia de mutação (uma realizando a união de *clusters* e outra escolhendo aleatoriamente entre a união de *clusters*, a divisão de *clusters* e a movimentação dos módulos) e variam entre a utilização ou não de um número máximo de *clusters* para participar de cada recombinação. Seus resultados mostram que a estratégia baseada na metaheurística ILS demonstrou ser uma heurística eficiente e eficaz para a resolução do PCMS.

2.3.3 Algoritmos genéticos interativos

Diferentemente das abordagens de clusterização totalmente automatizadas vistas anteriormente e baseados nos resultados obtidos por Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011), Bavota et al. (BAVOTA et al., 2012) definem um algoritmo genético interativo (AGI) mono-objetivo e multiobjetivo do PCMS permitindo que o *feedback* de desenvolvedores possa ser levado em conta durante o processo de clusterização. Partindo do mesmo princípio que a análise subjetiva de um desenvolvedor é bastante valiosa para o processo de clusterização, estratégia também observada em Mancoridis et al. (MANCORIDIS et al., 1999), o AGI permite que, após um número definido de gerações, a opinião de um desenvolvedor seja obtida tratando-a como uma restrição dentro do processo de geração de soluções. Por exemplo, um desenvolvedor pode indicar que determinados três módulos sempre sejam colocados em um mesmo *cluster*. A partir desse *feedback*, o AGI passará a respeitar essa restrição. Partindo desse princípio, duas abordagens são criadas para o AGI mono-objetivo do PCMS. A primeira abordagem seleciona a melhor solução dentre todas as gerações, após um número de gerações definido previamente, e solicita que o desenvolvedor opine se os dois módulos, também selecionados aleatoriamente, devem fazer parte de um mesmo *cluster* ou se eles devem ser separados em *clusters* distintos. A segunda versão do AGI mono-objetivo, também após um número de gerações definido previamente, seleciona um número de *clusters* considerados pequenos e solicita que o desenvolvedor opine a respeito desses *clusters*. Se o *cluster* pequeno em questão for unitário, o desenvolvedor deve decidir se ele continua como está ou se o módulo em questão deve ser migrado para outro *cluster*. Se o *cluster* em questão não for unitário, o desenvolvedor deve avaliar para cada par de componentes se devem permanecer juntos ou não. O AGI multiobjetivo é bastante similar ao mono-objetivo com a diferença que ao selecionar o indivíduo mais apto após um número especificado de gerações, outros aspectos são avaliados, diferentes do MQ e similares às métricas definidas por Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011). Os autores comparam seus resultados com os da abordagem multiobjetiva de Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011) e mostram que seus resultados são mais coerentes do ponto de vista de um desenvolvedor e indicam também que o MQ obtido na maioria das vezes é similar ou melhor do que as abordagens não interativas.

Também utilizando uma estratégia interativa, Simons et al. (SIMONS; PARMEE; GWYNLLYW, 2010) teorizam que a simetria/elegância de um projeto de software é um fator de qualidade relevante em um processo evolutivo e portanto pode ser incorporado em um processo interativo de computação evolutiva. Segundo os autores, essa simetria/elegância está diretamente relacionada à simetria e a uniformidade da distribuição entre os elementos de projeto de um sistema orientado a objetos (classes, métodos e atributos). Na abordagem utilizada por Simons et al. (SIMONS; PARMEE; GWYNLLYW, 2010) um *cluster* é uma classe e seus métodos e atributos são os módulos a serem clusterizados, diferentemente da maioria das abordagens, inclusive a desta dissertação, em que um *cluster* (namespaces em C# e *packages* em java) é um conjunto de módulos/arquivos (em C# arquivos .cs e em java arquivos .java). As métricas utilizadas para avaliar a elegância e simetria de um projeto de software adotadas são as seguintes:

1. *Numbers among classes elegance* (NAC): calcula-se primeiro a média e o desvio padrão do número de atributos por classe. Depois calcula-se a média e o desvio padrão do número de métodos por classe. A média dos dois desvios padrão calculados é o NAC. Quanto menor for este valor mais simétrica é a "aparência" dos métodos e atributos de uma classe;

2. *External couples elegance* (EC): é o desvio padrão do número de dependências entre métodos de classes distintas. Quanto menor for o EC, mais uniforme será a distribuição das dependências entre métodos de classes distintas;

3. *Internal uses elegance* (IU): para cada classe o número de usos "internos" é calculado (um uso interno se dá quando um método de uma classe acessa um atributo dessa classe). A média de usos internos por classe e seu desvio padrão são calculados. O IU é o desvio padrão. Quanto menor for o IU, mais uniforme será a distribuição dos usos internos de atributos por métodos de uma classe; e

4. *Attributes to methods ratio* (ATMR): é o desvio padrão da razão entre o número de atributos e métodos internamente às classes. Quando menor o valor do ATMR, mais simétrico é a relação entre atributos e métodos de uma classe.

Os experimentos com intuito de validar a teoria de simetria/elegância foram realizados em três problemas de *design* de software e seus resultados mostram que pelo menos três das métricas (NAC, EC, ATMR) estão em conformidade com a análise manual de arquitetos/desenvolvedores de software.

2.3.4 Algoritmos genéticos multiobjetivos

Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011) apresentam uma formulação multiobjetiva para o PCMS. Segundo os autores, a formulação mono-objetiva compara duas

grandezas (acoplamento e coesão) que estarão na maioria das vezes em conflito já que para se conseguir um baixo acoplamento naturalmente a coesão aumenta e vice-versa, o que pode resultar em soluções subótimas para o PCMS. Sendo assim, Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011) utilizam as fronteiras de Pareto para balancear objetivos distintos que podem entrar em conflito quando analisados de forma única. As fronteiras de Pareto são formuladas de acordo com a seguinte equação:

$$F(\overline{x_1}) > F(\overline{x_2}) \Leftrightarrow \forall i \cdot f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \wedge \exists i \cdot f_i(\overline{x_1}) > f_i(\overline{x_2}). \tag{2.7}$$

Para a solução candidata $\overline{x}$ o valor da função objetivo $F(\overline{x})$ é definido em termos dos valores de cada função constituinte f_i para $\overline{x}$. Assim, a solução $\overline{x_1}$ é melhor do que a solução $\overline{x_2}$ se ela é melhor de acordo com pelo menos uma das funções objetivo definidas e não é pior do que nenhuma outra. Nesse caso, é dito que a solução $\overline{x_1}$ é dominante em relação à solução $\overline{x_2}$. Alternativamente, se nenhum elemento de um conjunto $\overline{X}$ dominar alguma solução $\overline{x}$, então $\overline{x}$ não é "dominada" por $\overline{X}$. Esse estudo utiliza dois tipos de objetivos para a formulação multiobjetiva do PCMS: o *maximizing cluster approach* (MCA) e o *equal size cluster aproach* (ECA):

- *Maximizing cluster approach* (MCA): o MCA define os seguintes objetivos: maximizar a soma de arestas internas de um *cluster*; minimizar a soma de arestas externas de um *cluster*; maximizar o número de *clusters*; maximizar o MQ e minimizar o número de *clusters* unitários. O objetivo dessa métrica é obter o máximo de coesão, o mínimo de acoplamento e evitar a criação de *clusters* unitários já que, segundo o autor, experiência e intuição em relação a clusterização indicam que *clusters* unitários não fazem parte de softwares bem modularizados. A maximização do MQ também é incluída nessa análise multiobjetivo demonstrando a principal característica desse tipo de abordagem que permite a inclusão de novas funções objetivos a serem otimizadas;

- *Equal size cluster approach* (ECA): o ECA tem o objetivo de criar *clusters* que possuam aproximadamente o mesmo número de módulos, evitando *clusters* unitários ou *clusters* muito grandes que englobam muitos módulos. Essa uniformidade de distribuição de módulos ao longo dos *clusters* assemelha-se ao princípio de organização/elegância utilizado nos experimentos de Simons et al. (SIMONS; PARMEE; GWYNLLYW, 2010). Os objetivos do ECA são: maximizar a soma de arestas internas de todos os *clusters*; minimizar a soma de arestas externas de todos os *clusters*; maximizar o número de *clusters*; maximizar o MQ; minimizar a diferença entre o *cluster* com mais módulos e o *cluster* com menos módulos.

Essas duas formulações multiobjetivas foram implementadas utilizado o algoritmo evolucionário multiobjetivo *two-archive* de Praditwong e Yao (PRADITWONG; YAO, 2006) e em

seus experimentos Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011) comparam um *random restart hill climbing* utilizando uma estratégia mono-objetiva por meio do MQ e dois algoritmos genéticos multiobjetivos que utilizam respectivamente as formulações ECA e MCA (as formulações ECA e MCA superam as formulações mono-objetivas). Em seus experimentos foram utilizadas 17 instâncias reais ponderadas e não ponderadas com número de módulos variando de 20 a 100.

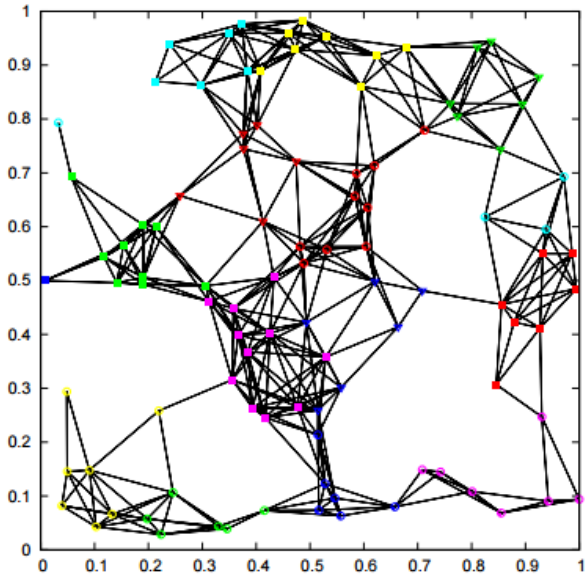
Em relação aos algoritmos genéticos, a codificação utilizada é a mesma de Mancoridis et al. (MANCORIDIS et al., 1999), ou seja, o método de codificação GNE é utilizado com uma taxa de mutação de $0.004 \times \log_2 n$, uma taxa de recombinação de 80% para populações de 100 indivíduos ou menos e uma taxa de 100% de recombinação para populações com mais de 100 indivíduos. O tamanho da população utilizada é $10 \times n$ sendo n o número de módulos do software a ser clusterizado e o número de gerações é de $200 \times n$. Em relação ao algoritmo de *hill climbing*, é utilizada a mesma configuração da ferramenta *Bunch* (MANCORIDIS et al., 1999). Seus resultados experimentais mostram que o *hill climbing* para grafos não ponderados obteve maiores valores de MQ do que a formulação MCA. No entanto, para grafos ponderados a formulação MCA gerou melhores resultados do que o *hill climbing*. Em relação ao ECA, os resultados para grafos não ponderados foram inconclusivos. No entanto, para grafos ponderados, o ECA foi estatisticamente superior em seis das sete instâncias. Finalmente, em uma comparação entre o ECA e o MCA, os resultados indicam que a formulação ECA superou o MCA em todas as instâncias e em apenas duas os resultados não foram estatisticamente significantes. Esses resultados foram confirmados por Can e Srinivasulu (CAN; SRINIVASULU, 2012), que replicaram esse estudo utilizando exatamente as mesmas instâncias.

2.3.5 Algoritmos genéticos de chaves aleatórias viciadas

Apesar de não ter sido encontrada uma aplicação do BRKGA especificamente para o PCMS, a técnica já foi utilizada para outros problemas de clusterização como pode ser visto em Morán-Mirabal et al. (MORÁN-MIRABAL et al., 2013). Nessa aplicação do BRKGA, o problema a ser resolvido é o *handover minimization problem* (HMP). Um *handover* em telecomunicação de redes celulares é a troca da conexão de um dispositivo móvel com sua respectiva torre de celular (*base stations*). Sempre que uma troca dessas ocorre, existe a probabilidade de uma ligação em andamento ser terminada, gerando uma experiência de uso ruim para o usuário impactado pela falha do *handover*. Complementando, as torres de celular conectam-se a uma *radio network controller* (RNC) que, entre outras responsabilidades, gerencia os *handovers*. Sempre que um *handover* ocorre entre torres de celular atribuídas a diferentes RNC, a probabilidade da falha do *handover* é maior do que se as torres estivessem atribuídas a uma mesma RNC. Sendo assim, o objetivo da aplicação do BRKGA para esse problema é atribuir torres de celular a RNC de modo que *handovers* entre torres seja minimizado sem que a quantidade limite de torres atribuídas a RNC seja ultrapassada. Esse problema também é conhecido como *node*

capacitated graph partioning problem (FERREIRA et al., 1998) que, assim como o PCMS, é um dos tipos de problemas de particionamento de grafos. A Figura 15 ilustra uma possível solução para o HMP.

Figura 15: Solução do HMP para 15 RNC e 100 torres de celular. Os nós coloridos representam as RNC (MORÁN-MIRABAL et al., 2013).



Ao se utilizar a estratégia BRKGA, é necessário definir os componentes livres da metaheurística específicos para o problema em questão, no caso: um codificador, um decodificador e uma função objetivo. Abaixo, uma breve descrição dos componentes do BRKGA conforme o trabalho proposto por Morán-Mirabal et al. (MORÁN-MIRABAL et al., 2013):

1. Codificador: as soluções para o HMP são codificadas como um vetor X de $2 \times |N|$ chaves aleatórias sendo N o conjunto de torres de celular. As primeiras $|N|$ posições indicam a ordem em que as torres de celular serão visitadas para que sejam atribuídas a RNC e as últimas $|N|$ chaves indicam quais RNC serão utilizadas na atribuição. Estratégia similar de codificação foi utilizada nesta dissertação como pode ser visto para o codificador CB apresentado na Seção 3.3;

2. Decodificador: para decodificar o vetor de $2 \times |N|$ chaves aleatórias e atribuir as torres de celular às RNC os seguintes passos devem ser realizados:

   a) Inicialização: inicializar um conjunto vazio de torres de celular, denominado T, e inicializar um conjunto vazio de torres de celular para cada RNC r existente;

   b) Ordenação: ordene crescentemente as primeiras $|N|$ posições do vetor de chaves aleatórias para produzir uma permutação das torres de celular. Essa ordenação definirá a ordem que a tentativa de atribuição às RNC será realizada;

   c) Atribuição: considerando a ordem definida no Passo b), o RNC r a ser atribuído para a torre de celular i é $r = \lceil X[|N| + i] \times |R| \rceil$ sendo R o conjunto de RNC existentes

se essa RNC ainda tiver capacidade para receber a respectiva torre de celular. Por exemplo, um vetor $X = [0{,}45;\ 0{,}33;\ 0{,}56;\ 0{,}78;\ 0{,}34;\ 0{,}001]$ de seis posições indica que existem três torres de celular a serem atribuídas. Se existirem duas RNC, isto significa que a torre de celular $X[0]$ representada pela chave aleatória $0{,}45$ vai tentar ser atribuída a RNC 2 pois $\lceil X[3+0] \times 2 \rceil = \lceil 0{,}78 \times 2 \rceil = \lceil 1{,}56 \rceil = 2$. Se a RNC escolhida não tiver capacidade para receber a torre de celular, essa é atribuída a lista T definida no Passo a);

d) Atribuição final: considerando as torres de celular em T, seguindo a ordem definida no Passo b), se existir um conjunto de RNC que tenha capacidade de receber a torre de celular em questão, considere esse conjunto de RNC. Senão, considere todas as RNC. Atribua a torre de celular a RNC que já possui as torres de celular que possuam um máximo número de *handovers* com a RNC a ser atribuída. Se por acaso nenhuma RNC tiver capacidade para receber uma torre de celular uma atribuição inviável serão realizada e essa solução será penalizada com um alto custo de *handover*;

e) Busca Local: aplique uma busca local iterando pelas torres de celular definidas no Passo b). Itere pelas RNC em ordem crescente de seus índices. Se ao mover a torre de celular i da sua RNC atual para outra RNC que tenha capacidade de recebê-la diminuir o número total de *handovers*, logo essa movimentação deve ser realizada. Esse procedimento deve ser executado até que nenhuma movimentação possa ser realizada; e

f) Ajuste dos cromossomos: se algum movimento for realizado pelo passo anterior, a segunda metade do vetor de chaves aleatórias deve ser modificada para refletir a nova realidade. Por exemplo, se a torre de celular i foi inicialmente atribuída a RNC $j = \lceil X[|N| + i] \rceil$ e foi movimentada para a RNC k durante a busca local, o cromossomo deve ser ajustado da seguinte forma: $X[|N| + i] \leftarrow X[|N| + i] + (k - j) \times |R|^{-1}$.

Em Stefanello et al. (STEFANELLO et al., 2015) foi utilizado um BRKGA para minimizar a largura de banda necessária para atender os requisitos de qualidade de usuários de computação na nuvem. A solução obtida pelo BRKGA indica, respeitando as restrições do problema, em quais *data centers* as máquinas virtuais devem ser instaladas. Os autores utilizaram um tipo de codificador em que o vetor de chaves aleatórias é usado para definir a ordem na qual as máquinas virtuais serão visitadas para serem alocadas nos *data centers*. Essa estratégia é exatamente a mesma do codificador CA definido na Seção 3.1 desta dissertação e é similar à estratégia definida ao codificador CB. Em relação aos decodificadores definidos em (STEFANELLO et al., 2015), é interessante ressaltar o decodificador "D1" que utiliza uma estratégia *best fit* para alocar as máquinas virtuais nos *data centers* assim como o decodificador BF desta dissertação, que aloca os módulos no *cluster* que maximiza o MQ da solução em questão.

2.3.6 Hiper-heurísticas

Diferentemente das abordagens anteriores, Kumari et al. (KUMARI; SRINIVAS; GUPTA, 2013) utilizam uma hiper-heurística para resolver o PCMS. Segundo Cowling et al. (COWLING; KENDALL; SOUBEIGA, 2001), hiper-heurísticas gerenciam a escolha de qual heurística de baixo nível deve ser aplicada em um dado momento segundo a característica da heurística e do espaço de busca corrente a ser explorado. Sua principal diferença para uma metaheurística é que metaheurísticas trabalham diretamente no espaço de busca do problema a ser resolvido, enquanto que hiper-heurísticas mudam o foco buscando heurísticas para resolver o problema ao invés de procurar soluções para o problema em si (KUMARI; SRINIVAS; GUPTA, 2013). A hiper-heurística proposta pelos autores, chamada de MHypEA, é baseada no uso de algoritmos evolutivos multiobjetivos e procura escolher entre 12 heurísticas baseadas nos operadores de seleção, recombinação e mutação. O processo de escolha das heurísticas baseia-se no princípio de *reinforcement learning with adaptative weights*, ou seja, a cada escolha de uma heurística, sua eficácia é avaliada e um peso é atribuído a ela de acordo com o que irá influenciar sua escolha nos próximos processos de seleção das heurísticas. Quanto maior sua eficácia, maior será a probabilidade da heurística ser novamente escolhida. Abaixo, um resumo dos operadores genéticos que dão origem às 12 heurísticas criadas:

1. Operadores de seleção: operador de seleção *rand* em que ambos os pais são selecionados de forma aleatória da população; operador de seleção *rand-to-best* em que um pai é selecionado aleatoriamente de toda a população e outro pai é selecionado de um subconjunto elite da população;

2. Operadores de recombinação: operador *uniform crossover*, o qual seleciona aleatoriamente os genes dos pais para que a prole possa ser formada; operador *hybrid crossover* 1 (hc1), uma combinação do *single-point-crossover* com o *uniform crossover*; operador *hybrid crossover* 2 (hc2), uma combinação do *uniform crossover* com o *two-point crossover*; e

3. Operadores de mutação: operador *copy* em que dois genes são selecionados aleatoriamente e o segundo gene é copiado para o primeiro; operador *exchange* em que dois genes selecionados aleatoriamente trocam de posição.

A eficiência e a eficácia da heurística MHypEA foi avaliada em seis instâncias reais da literatura variando-se o tamanho da instância de 20 a 174 módulos e de 57 a 360 arestas. Seus resultados foram comparados com o algoritmo genético multiobjetivo *two-archive* de Praditwong et al. (PRADITWONG; HARMAN; YAO, 2011) e indicam uma leve superioridade da heurística MHypEA em termos de eficácia (menos de 1% de diferença) e em termos de eficiência consome aproximadamente um vigésimo do tempo computacional do *two-archive*.

2.3.7 Considerações finais

Por último, a Tabela 1 apresenta uma relação com 18 importantes trabalhos que abordam o PCMS. As primeiras duas colunas apresentam, em ordem cronológica, o ano e o título do trabalho, a terceira coluna apresenta as abordagens utilizadas para resolver o PCMS, entre elas, é possível citar: algoritmos genéticos (AG), busca local do tipo *Hill Climbing* (HC), a metaheurística *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003), o método NSGA-II (SRINIVAS; DEB, 1994) e métodos exatos. A quarta coluna indica se a função objetivo é mono-objetiva ou multiobjetiva e, entre parenteses, os nomes das funções objetivos utilizadas. As duas últimas colunas apresentam, respectivamente, a quantidade de instâncias utilizadas nos experimentos e a quantidade de módulos da maior instância utilizada no respectivo trabalho.

O próximo capítulo apresenta uma proposta de resolução para o problema de clusterização de módulos de software (PCMS) baseada em algoritmos genéticos de chaves aleatórias viciadas (BRKGA), a partir da definição de um conjunto de codificadores e decodificadores.

Tabela 1: Resumo dos principais trabalhos sobre o PCMS.

Referência		Abordagens	Função objetivo	# inst.	$> n$
1998	Using automatic clustering to produce high-level system organizations of source code (MANCORIDIS et al., 1998)	Exato do tipo busca exaustiva	Mono-objetivo (MQ)	4	153
1999	Bunch: a clustering tool for the recovery and maintenance of software system structures (MANCORIDIS et al., 1999)	AG, busca local (HC) e exato do tipo busca exaustiva	Mono-objetivo (MQ)	1	40
1999	Automatic clustering of software systems using a genetic algorithm (DOVAL; MANCORIDIS; MITCHELL, 1999)	AG	Mono-objetivo (MQ)	1	20
2003	A heuristic search approach to solving the software clustering problem (MITCHELL, 2003)	Exato do tipo busca exaustiva, busca local (HC) e AG	Mono-objetivo (MQ)	8	955
2003	A multiple hill climbing approach to software module clustering (MAHDAVI; HARMAN; HIERONS, 2003)	Busca local (HC) com múltiplas partidas	Mono-objetivo (MQ)	19	413
2003	Efficient evolutionary algorithms for the clustering problem in directed graphs (DIAS; OCHI, 2003)	AG	Mono-objetivo (MQ)	22	500
2005	An empirical study of the robustness of two module clustering functions (HARMAN, 2005)	Busca local (HC)	Mono-objetivo (MQ e EVM)	12	174
2011	Solving software module clustering problem by evolutionary algorithms (PRADITWONG, 2011)	AG	Mono-objetivo (MQ)	17	198
2011	Agrupamento de sistemas orientados a objetos com metaheurísticas evolutivas (BOTELHO; SEMAAN; OCHI, 2011)	AG	Mono-objetivo (MQ)	10	100
2011	Software module clustering as a multi-objective search problem (PRADITWONG; HARMAN; YAO, 2011)	AG, busca local (HC)	Multiobjetivo (MCA e ECA)	17	198
2012	Multi-objective approach for software module clustering (CAN; SRINIVASULU, 2012)	AG	Multiobjetivo (MCA, ECA e MQ)	17	198

Tabela 1: (continuação)

Referência		Abordagens	Função objetivo	# inst.	$> n$
2012	An analysis of the effects of composite objectives in multiobjective software module clustering (BARROS, 2012)	NSGA-II	Multiobjetivo (MQ e EVM)	14	195
2012	Putting the developer in-the-Loop: an interactive GA for software re-modularization (BAVOTA et al., 2012)	AG	Mono-objetivo (MQ) e Multiobjetivo (MQ)	2	297
2013	Software module clustering using a hyper-heuristic based multi-objective genetic algorithm (KUMARI; SRINIVAS; GUPTA, 2013)	Hiper-heurística, AG	Mono-objetivo (MQ)	6	174
2013	Mixed-Integer Linear Programming Formulations for the Software Clustering Problem (KÖHLER; FAMPA; ARAUJO, 2013)	Método exato do tipo programação linear inteira mista	Mono-objetivo (MQ)	45	62
2014	ILS for the software module clustering problem (PINTO; ALVIM; BARROS, 2014)	ILS	Mono-objetivo (MQ)	40	483
2014	Uma heurística baseada em busca local iterada para o problema de clusterização de módulos de software (PINTO, 2014)	ILS, AG e busca local (HC)	Mono-objetivo (MQ)	46	483
2015	Column generation aproaches for the software clustering problem (KRAMER et al., 2014)	Método exato do tipo *Branch and Price*	Mono-objetivo (MQ)	55	55

3 Algoritmos Genéticos de Chaves Aleatórias Viciadas para o Problema de Clusterização de Módulos de Software

O objetivo deste capítulo é apresentar a proposta de solução desta dissertação que é uma heurística baseada em algoritmos genéticos de chaves aleatórias viciadas (BRKGA) para o problema de clusterização de módulos de software (PCMS).

Quando desenvolvendo heurísticas baseadas em metaheurísticas, é necessário fazer escolhas específicas para os componentes livres da metaheurística. No caso do método BRKGA, é necessário definir um codificador, um decodificador, uma função de aptidão e os valores dos parâmetros de configuração do BRKGA. A função de aptidão no presente trabalho é a função MQ (ver Seção 1.1). A seguir, descrevem-se dois codificadores e cinco decodificadores para o PCMS. A Tabela 2 mostra quais os parâmetros do *framework* que necessitam de configuração sugerindo, na coluna "Valor Recomendado", alguns valores obtidos empiricamente na aplicação do BRKGA proposta por Morán-Mirabal et al. (MORÁN-MIRABAL et al., 2013). Daqui em diante, o algoritmo resultante será chamado de BRKGA_PCMS.

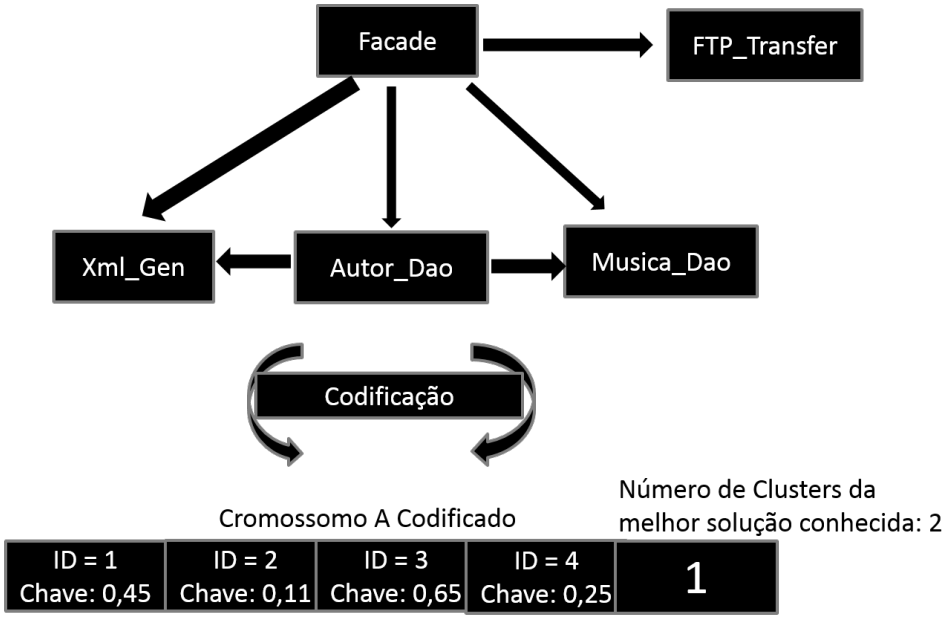
Tabela 2: Parâmetros de configuração do BRKGA.

Parâmetro	Descrição	Valor Recomendado
p	tamanho da população	$p = an$ onde a é uma constante e n é o tamanho do cromossomo
p_e	tamanho da população elite	$0.10p \leq p_e \leq 0.25p$
p_m	tamanho da população mutante	$0.10p \leq p_m \leq 0.30p$
p_p	probabilidade de uma prole herdar um alelo elite	$0.50p \leq p_p \leq 0.80p$

3.1 Codificador CA

Considere a representação de solução onde o cromossomo é um vetor de $n+1$ elementos. Nesta representação, uma vez ordenado o vetor de cromossomos, os n primeiros genes são números aleatórios gerados no intervalo real de [0,1] e determinam a ordem na qual os módulos serão considerados no processo de construção de uma solução (clusterização). O $n+1$-ésimo gene "carrega" a informação do número m de *clusters* que será considerado para a clusterização. Neste codificador, m é um número inteiro aleatório calculado no intervalo de 20% a 40% de n. Este intervalo foi obtido por meio da análise das melhores soluções obtidas por Pinto (PINTO, 2014). A Figura 16 exemplifica este processo de codificação.

Figura 16: Exemplo de Codificação do BRKGA_PCMS. O procedimento BRKGA_PCMS recebe como entrada um grafo MDG. Os retângulos representam os módulos de um software e as setas representam uma referência direta entre dois módulos. Por exemplo, na figura, o módulo "Façade" utiliza alguma funcionalidade do módulo "MUSICA_DAO" e por isso possui uma referência direta para o mesmo. O processo de codificação transforma este grafo em um vetor de chaves aleatórias onde cada módulo possui associado uma chave aleatória.



3.2 Decodificadores BF, WF, LBF e FF

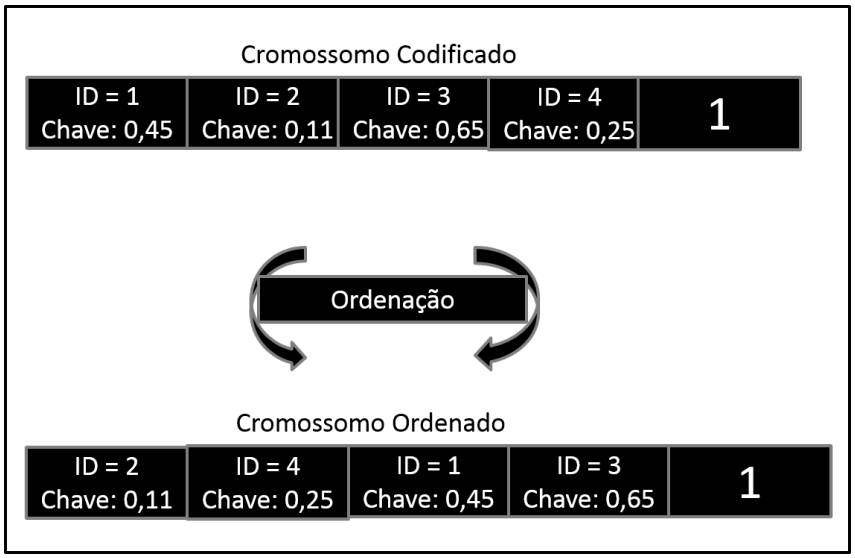
Um decodificador é responsável por, a partir de um vetor de chaves aleatórias, gerar uma possível solução para o BRKGA_PCMS. A seguir, são defindos quatro decodificadores que serão usados em conjunto com o Codificador CA definido na seção anterior.

1. Inicialização: neste passo, m *clusters* são criados, inicialmente vazios. Cada *cluster* é representado por uma lista de módulos. Assume-se que os *clusters* são indexados $1, 2, \ldots, m$;

2. Ordenar os módulos: ordene de forma crescente as n chaves aleatórias do vetor para produzir uma permutação de módulos. Suponha que cada gene do cromossomo possui, além de sua chave aleatória, um identificador responsável por "guardar" sua posição original no vetor. A Figura 17 ilustra a representação do cromossomo antes e depois da ordenação. Em cada gene, $chave$ guarda a informação da chave aleatória e ID a informação da identificação do módulo. O índice do vetor indica a ordem de visitação dos módulos, identificados por ID, usada no processo de atribuição de *clusters* aos módulos. Percebe-se que a última posição do vetor, que indica o número de *clusters* a ser utilizado, não é influenciada pela ordenação;

3. Atribuição inicial dos módulos: para cada *cluster* aberto no passo 1, itere pelos módulos

Figura 17: Exemplo de ordenação das chaves aleatórias no BRKGA_PCMS.



em ordem crescente de seus índices e selecione o primeiro módulo que ainda não possui *cluster*, alocando-o neste *cluster* em conjunto com um percentual p_v de seus vizinhos (outros módulos que possuem dependência com o módulo em questão);

4. Atribuição dos módulos restantes: respeitando a ordenação definida no passo 2, visite os módulos que não foram clusterizados no passo anterior. Ao visitar um módulo, é possível escolher uma das quatro estratégias distintas em relação à sua alocação em um *cluster*:

   a) Decodificador BF - Estratégia Best-Fit: o módulo é colocado no *cluster* que maximiza o MQ;

   b) Decodificador WF - Estratégia Worst-Fit: o módulo é colocado no *cluster* que minimiza o MQ;

   c) Decodificador LBF - Estratégia Least-Best-Fit: o módulo é colocado no *cluster* que menos melhora o MQ; e

   d) Decodificador FF - Estratégia First-Fit: o módulo é colocado no primeiro *cluster*, visitado em ordem do seu índice, que melhora o MQ.

5. Atribuição dos últimos módulos: independente da estratégia escolhida acima, caso ainda existam módulos sem nenhum *cluster*, cada um deles é colocado no *cluster* que maximiza o MQ. Se não existir tal *cluster* (qualquer clusterização somente piora o MQ), um novo *cluster* é criado somente para este módulo.

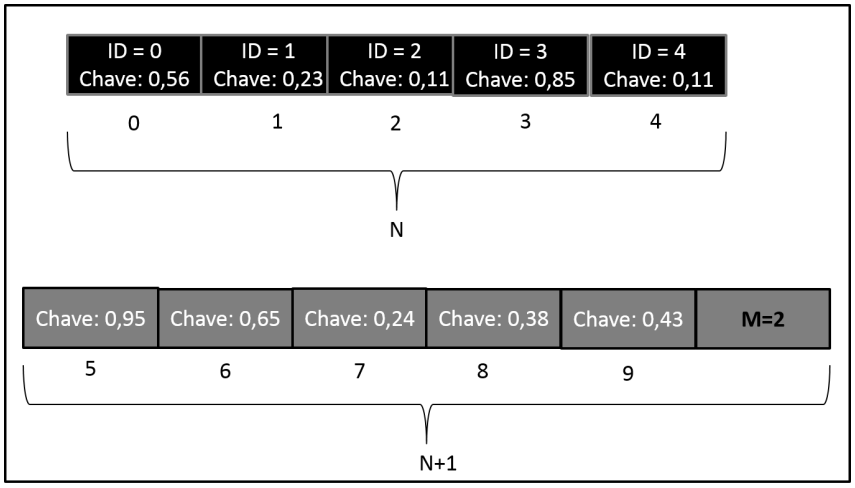
3.3 Codificador CB

Considere a representação de solução onde o cromossomo é um vetor de $2n+1$ elementos. Nesta representação, os $2n$ primeiros genes são números aleatórios gerados no intervalo real

de [0,1]. Os primeiros n genes determinam a ordem na qual os módulos serão considerados no processo de construção de uma solução (clusterização). Os últimos n elementos determinam qual estratégia de clusterização será utilizada (a seção a seguir, do decodificador, detalha como esta informação é utilizada para realizar a clusterização).

O $2n + 1$-ésimo gene "carrega" a informação do número m de *clusters* que será considerado para a clusterização. Neste codificador, m é um número inteiro aleatório gerado no intervalo de 20% a 40% de n. A Figura 18 apresenta um exemplo deste tipo de codificação.

Figura 18: Cromossomo de tamanho $2n + 1$ do BRKGA_PCMS codificado por CB.



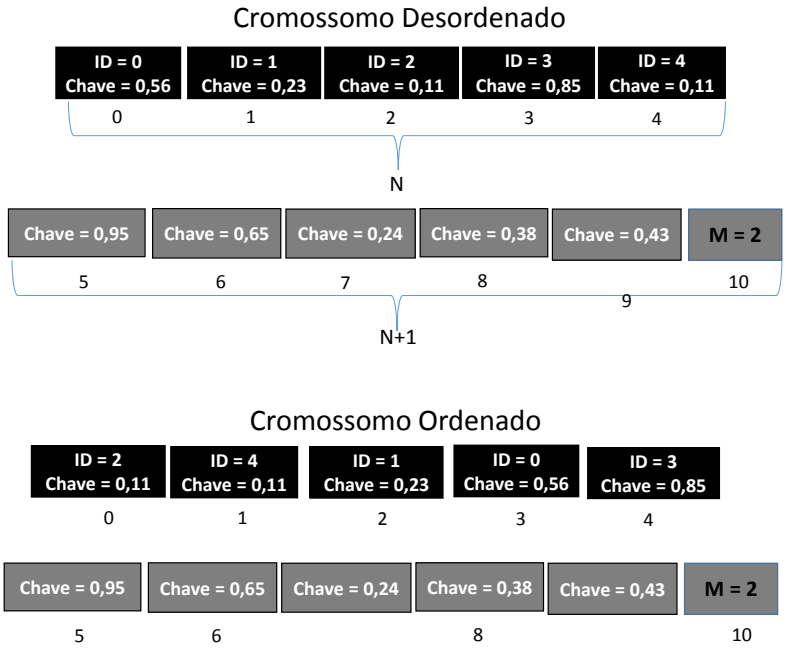
3.4 Decodificador LBF/BF

1. Inicialização: neste passo, m *clusters* são criados, inicialmente vazios. Cada *cluster* é representado por uma lista de módulos. Assume-se que os *clusters* são indexados $1, 2, \ldots, m$;

2. Ordenar os módulos: ordene de forma crescente as n primeiras chaves aleatórias do vetor para produzir uma permutação de módulos. Suponha que cada gene do cromossomo possui, além de sua chave aleatória, um identificador responsável por "guardar" sua posição original no vetor. A Figura 19 ilustra a representação do cromossomo antes e depois da ordenação. Em cada gene, das n primeiras posições, $chave$ guarda a informação da chave aleatória e ID a informação da identificação do módulo. Embaixo de cada gene, tem-se a informação do índice do vetor. O índice do vetor indica a ordem de visitação dos módulos, identificados por ID, usada no processo de atribuição de *clusters* aos módulos. Neste tipo de representação, os $n + 1$ últimos elementos não sofrem influência deste processo de ordenação;

3. Atribuição inicial dos módulos: para cada *cluster* aberto no passo 1 selecione o primeiro módulo que ainda não possui *cluster*, alocando-o neste *cluster* em conjunto com um percentual p_v de seus vizinhos (outros módulos que possuem dependência com o módulo em questão);

Figura 19: Cromossomo de tamanho $2n + 1$ do BRKGA_PCMS.



4. Atribuição dos módulos restantes: respeitando a ordenação definida no passo 2, visite os módulos que ainda não foram clusterizados no passo anterior. Ao visitar um módulo, é possível utilizar duas estratégias distintas em relação à sua alocação em um *cluster*:

   a) Estratégia Least-Best-Fit: se ao visitar o módulo i, o módulo indicado na posição do vetor $n + i$ for um alelo de valor maior que $0{,}5$, utilize a estratégia Least-Best-Fit, conforme definida na Seção 3.2;

   b) Estratégia Best-Fit: se ao visitar o módulo i, o módulo indicado na posição do vetor $n+i$ for um alelo de valor menor ou igual a $0{,}5$, utilize a estratégia Best-Fit, conforme definida na Seção 3.2.

5. Atribuição dos últimos módulos: independente da estratégia anteriormente escolhida, caso ainda existam módulos sem nenhum *cluster*, cada um deles é colocado no *cluster* que maximiza o MQ. Se não existir tal *cluster* (qualquer clusterização somente piora o MQ), um novo *cluster* é criado somente para este módulo.

3.5 Busca local

Ao final da execução do BRKGA_PCMS, o algoritmo retorna a melhor solução entre todas as gerações. Em seguida, é executado um procedimento de busca local na tentativa de melhorar esta solução. A vizinhança $N(s)$ de uma dada solução s é definida como o conjunto de soluções obtidas por meio da movimentação de um módulo de um *cluster* para outro *cluster*

e foi implementada da seguinte forma: para cada *cluster*, visitado em ordem crescente de seus índices, considera-se cada módulo, visitado em ordem crescente de seus índices, e calcula-se o MQ da solução obtida movendo-se o módulo corrente do seu *cluster* para cada outro *cluster*. O movimento que obtiver o melhor MQ é selecionado. Este processo deve ser repetido até que nenhum movimento melhore o MQ da solução. Note que este procedimento de busca local não permite aumentar a quantidade de *clusters* da solução. Entretanto, se o último módulo de um *cluster* for transferido para outro *cluster*, o número de *clusters* da solução é reduzido em uma unidade.

A implementação deste procedimento foi feita de maneira que, quando movendo um módulo de um *cluster* para outro, não é necessário recalcular o CF de todos os *clusters*, sendo necessário somente recalcular os CF dos *clusters* impactados pela movimentação. Esta otimização no cálculo da atualização do valor do MQ é conhecida na literatura como ITurboMQ (MITCHELL, 2002), tendo sido apresentada anteriormente na Seção 2.1.3.

3.6 Considerações finais

Este capítulo apresentou diversas escolhas para os parâmetros e os componentes do BRKGA, quando aplicado ao PCMS. Com o objetivo de eleger a melhor versão para o algoritmo proposto, essas diversas escolhas para os componentes e parâmetros do BRKGA serão avaliadas em experimentos computacionais descritos no próximo capítulo.

4 Experimentos computacionais

Nesta seção descrevem-se os experimentos computacionais realizados aplicando-se a heurística BRKGA_PCMS em um conjunto de problemas testes.

Todos os experimentos foram realizados em uma máquina com processador Intel Xeon CPU E5-1607 com 32GB de RAM disponível e dedicação exclusiva. Todos os algoritmos implementados nesta dissertação foram codificados na linguagem C# e compilados com o .NET Framework 4.5. Para a geração de números pseudo-aleatórios utilizou-se o gerador fornecido pelo .NET Framework, baseado no *Subtractive Random Number Generator Algorithm* (KNUTH, 1981).

4.1 Questões de pesquisa

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

1. QP1: Eficácia como critério de avaliação: a heurística baseada no BRKGA produz soluções com maior valor de MQ do que as soluções obtidas por outros métodos da literatura?

   A média dos valores alcançados pela função objetivo MQ aplicando-se a heurística baseada no BRKGA foi comparada com os valores de MQ de 113 instâncias encontradas na literatura, incluindo resultados obtidos por quatro métodos genéticos e um método exato.

2. QP2: Eficiência como critério de avaliação: a heurística baseada em BRKGA produz soluções consumindo menos tempo de execução do que o tempo gasto para produzir soluções para o mesmo problema por outros métodos da literatura?

   O tempo de execução da heurística baseada no BRKGA foi comparado com o tempo de processamento de 113 instâncias encontradas na literatura, incluindo resultados obtidos por quatro métodos genéticos e um método exato.

4.2 Instâncias de teste

Para avaliar o desempenho do método proposto, utilizou-se um conjunto de 113 instâncias divididas em três classes, conforme exibido na Tabela 3.

A Tabela 4 relaciona todas as instâncias de teste. A primeira coluna (“Instância”) exibe o identificador da instância; a segunda coluna (“# Mod.”) mostra a quantidade de módulos da instância; a terceira coluna (“# Dep.”) exibe a quantidade de dependências entre seus módulos; a próxima coluna (“Peso”) indica se as dependências possuem peso (’S’) ou não (’N’); a coluna

Tabela 3: Instâncias classificadas por quantidade de módulos.

Classe	Quantidade de módulos	Quantidade de Instâncias
Pequena	Até 100 módulos	70
Média	De 101 até 200 módulos	23
Grande	Acima de 200 módulos	20

z^* indica o valor ótimo provado ou (-) para indicar que não se conhece o valor ótimo; por fim, a última coluna "Referência" relaciona a referência que possui o valor de MQ ótimo daquela instância em seus experimentos ou (-) quando o valor ótimo do MQ não tiver sido encontrado em nenhuma das referências estudadas. A maior instância ("apache_lucene_core") possui 738 módulos e 3.726 dependências.

Tabela 4: Conjunto de 113 instâncias de teste.

Instâncias pequenas (70)					
Instância	# Mod.	# Dep.	Peso	z^*	Referência
squid	2	2	S	-	-
small	6	5	N	1,8333	(KÖHLER; FAMPA; ARAUJO, 2013)
compiler	13	32	N	1,50649	(KÖHLER; FAMPA; ARAUJO, 2013)
lab4	15	18	N	3,4	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-ping	15	15	S	1	(KÖHLER; FAMPA; ARAUJO, 2013)
nss_ldap	15	16	S	1	(KÖHLER; FAMPA; ARAUJO, 2013)
jstl	15	20	N	-	-
nos	16	52	N	1,67748	(KÖHLER; FAMPA; ARAUJO, 2013)
lslayout	17	43	N	1,8613	(KÖHLER; FAMPA; ARAUJO, 2013)
boxer	18	29	N	3,1011	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-tftpd	18	23	S	1,14421	(KÖHLER; FAMPA; ARAUJO, 2013)
sharutils	19	36	S	2,54921	(KÖHLER; FAMPA; ARAUJO, 2013)
mtunis	20	57	N	2,31446	(KÖHLER; FAMPA; ARAUJO, 2013)
spdb	21	33	N	5,5897	(KÖHLER; FAMPA; ARAUJO, 2013)

Tabela 4: (continuação)

Instância	# Mod.	# Dep.	Peso	z^*	Referência
xtell	22	57	S	2,00523	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-inetd	24	25	S	1,3121	(KÖHLER; FAMPA; ARAUJO, 2013)
bunch	24	62	N	2,40602	(KÖHLER; FAMPA; ARAUJO, 2013)
ispell	24	103	N	2,3639	(KÖHLER; FAMPA; ARAUJO, 2013)
nanoxml	25	64	N	-	-
ciald	26	64	N	2,85291	(KÖHLER; FAMPA; ARAUJO, 2013)
jodamoney	26	102	N	-	-
bootp	27	75	S	2,19853	(KÖHLER; FAMPA; ARAUJO, 2013)
Modulizer	27	66	N	2,7579	(KÖHLER; FAMPA; ARAUJO, 2013)
jxlsreader	27	73	N	-	-
telnetd	28	81	S	1,8475	(KÖHLER; FAMPA; ARAUJO, 2013)
sysklogd-1	28	74	S	1,7105	(KÖHLER; FAMPA; ARAUJO, 2013)
crond	29	112	S	2,303	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-ftp	29	95	S	1,768	(KÖHLER; FAMPA; ARAUJO, 2013)
rcs	29	163	N	-	-
seemp	30	61	N	-	-
dhcpd-2	31	122	S	3,49437	(KÖHLER; FAMPA; ARAUJO, 2013)
cyrus-sasl	32	100	S	3,25182	(KÖHLER; FAMPA; ARAUJO, 2013)
tcsh	32	105	S	1,2141	(KÖHLER; FAMPA; ARAUJO, 2013)
micq	33	156	S	2,16799	(KÖHLER; FAMPA; ARAUJO, 2013)
apache	36	86	N	-	-
star	36	89	N	-	-
bison	37	179	N	-	-

Tabela 4: (continuação)

Instância	# Mod.	# Dep.	Peso	z^*	Referência
stunnel	38	97	S	2,52605	(KÖHLER; FAMPA; ARAUJO, 2013)
cia	38	636	N	-	-
minicom	40	257	S	2,57585	(KÖHLER; FAMPA; ARAUJO, 2013)
mailx	41	331	S	-	-
screen	41	289	S	-	-
dot	43	255	N	-	-
slang	45	242	S	-	-
slrn	45	323	S	-	-
net-tools	48	183	S	-	-
wu-ftpd-1	50	230	S	2,44373	(KÖHLER; FAMPA; ARAUJO, 2013)
joe	51	540	S	-	-
imapd-1	53	298	S	3,62499	(KÖHLER; FAMPA; ARAUJO, 2013)
wu-ftpd-3	54	278	S	-	-
udt-java	56	227	N	-	-
javaocr	58	155	N	-	-
dhcpd-1	59	571	S	-	-
icecast	60	650	S	2,7544	(KÖHLER; FAMPA; ARAUJO, 2013)
pfcda_base	60	197	N	-	-
servletapi	61	131	N	-	-
php	62	191	S	5,32421	(KÖHLER; FAMPA; ARAUJO, 2013)
bunch2	65	151	N	-	-
forms	68	270	N	-	-
jscatterplot	74	232	N	-	-
jxlscore	79	330	N	-	-
elm-2	81	683	S	-	-
jfluid	81	315	N	-	-
grappa	86	295	N	-	-
elm-1	88	941	S	-	-
gnupg	88	601	S	-	-
inn	90	624	S	-	-
bash	92	901	S	-	-
jpassword	96	361	N	-	-
bitchx	97	1653	S	-	-

Tabela 4: (continuação)

Instância	# Mod.	# Dep.	Peso	z^*	Referência
Instâncias médias (23)					
Instância	# Mod.	# Dep.	Peso	z^*	Referência
junit	100	274	N	-	-
xntp	111	729	S	-	-
acqCigna	114	188	N	-	-
bunch_2	116	364	N	-	-
xmldom_n	118	209	N	-	-
exim	118	1255	S	-	-
cia++	124	369	N	-	-
tinytim	129	564	N	-	-
mod_ssl	135	1095	S	-	-
jkaryoscope	136	460	N	-	-
ncurses_n	138	682	S	-	-
gae_plugin_core	139	375	N	-	-
lynx	148	1745	S	-	-
lucent	153	103	N	-	-
javacc	153	722	N	-	-
JavaGeom	171	1445	N	-	-
incl	174	360	N	-	-
jdendogram	177	583	N	-	-
xmlapi	182	413	N	-	-
jmetal	190	1137	N	-	-
dom4j	195	930	N	-	-
nmh	198	3262	S	-	-
pdf_renderer	199	629	N	-	-
Instâncias grandes (20)					
Instância	# Mod.	# Dep.	Peso	z^*	Referência
Jung_graph_model	207	603	N	-	-
jconsole	220	859	N	-	-
jung_visualization	221	919	N	-	-
pfcda_swing	252	885	N	-	-
jpassword2	269	1348	N	-	-
jml	270	1745	N	-	-
notelab-full	293	1349	N	-	-
poormans CMS	301	1118	N	-	-
log4j	305	1078	N	-	-
jtreeview	320	1057	N	-	-
bunchall	324	1339	N	-	-
jace	338	1524	N	-	-

Tabela 4: (continuação)

Instância	# Mod.	# Dep.	Peso	z^*	Referência
javaws	377	1403	N	-	-
swing	413	1513	N	-	-
lwjgl-2.8.4	453	1976	N	-	-
ping_libc	481	2854	S	-	-
y_base	556	2510	N	-	-
krb5	558	3793	S	-	-
apache_ant_taskdef	626	2421	N	-	-
apache_lucene_core	738	3726	N	-	-

As 113 instâncias de teste foram selecionadas em sua maioria de aplicações reais *open-source* com código-fonte disponível em Java. Para estas instâncias, 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". Esse formato identifica as dependências estáticas existentes entre classes da aplicação e entre classes e pacotes.

Desse conjunto de 113 instâncias selecionou-se um subgrupo (instâncias em negrito na Tabela 4) composto por cinco instâncias da classe "Pequena", quatro da classe "Média" e três com mais de 200 módulos. Este subgrupo foi utilizado no primeiro e segundo estudos experimentais. O terceiro estudo utilizou todas as instâncias de teste.

4.3 Comparação de métodos heurísticos iterativos não determinísticos

Conforme Taillard (TAILLARD, 2001), comparar dois (ou mais) métodos heurísticos baseados em princípios metaheurísticos é uma tarefa difícil que ainda não é resolvida de forma satisfatória. Uma característica desses métodos iterativos é que a qualidade da solução produzida tende a aumentar com o aumento do tempo de execução. Além disso, a maioria desses métodos são não-determinísticos, isto é, possuem componentes aleatórias. Isto significa que executar o mesmo método heurístico duas vezes pode resultar em duas soluções distintas.

A comparação entre dois métodos iterativos deve ser feita considerando-se a qualidade da solução produzida e o esforço computacional, que tradicionalmente é o tempo de execução em uma dada máquina. Entretanto, essas medidas não são precisas. O tempo de processamento depende fortemente do estilo de programação, do compilador, das opções de compilação usadas, etc. Em relação à qualidade da solução, a comparação de métodos não determinísticos

também não é simples uma vez que a qualidade da solução não depende somente do esforço computacional, mas também da semente utilizada pelo gerador de números pseudo-aleatórios.

A existência de tais componentes aleatórias exige que se realize múltiplas execuções de um dado algoritmo e que se use estatísticas descritivas (como média, desvio padrão e mediana, por exemplo) para representação dessas medidas. Assim, dados dois métodos heurísticos, A e B, e a qualidade das soluções obtidas pela execução de n_a vezes A e de n_b vezes B, deseja-se saber se A é melhor do que B. Quando tais amostras são pequenas e suas distribuições desconhecidas, o que, em geral, acontece quando comparando dois métodos heurísticos com componentes aleatórias, não é possível realizar uma comparação dos dois métodos usando métodos paramétricos de inferência estatística, que são aplicáveis a dados distribuídos segundo uma curva normal.

Em tais casos, em que não se garante normalidade e homocedasticidade das amostras independentes, é possível utilizar o teste estatístico não paramétrico de Wilcoxon-Mann-Whitney (FELTOVICH, 2003) para verificar se a diferença entre as médias (ou medianas) dos valores das soluções obtidas pelos dois métodos foi, de fato, devido à superioridade de algum deles ou simplesmente devido a componentes aleatórias dos métodos. De forma bem simplificada, o teste de Wilcoxon-Mann-Whitney funciona da seguinte maneira: a hipótese nula é que não existe diferença entre as médias encontradas pelos dois métodos. Se a probabilidade que essa hipótese seja verdadeira é menor do que um dado nível de significância α, considerando as n_a e n_b soluções obtidas, então a hipótese nula é rejeitada e a hipótese alternativa é aceita. A hipótese alternativa indica que um método é melhor do que o outro. A identificação de qual método é melhor pode ser feita pela comparação direta dos valores das médias (ou medianas) dos n_a/n_b valores observados.

Neste capítulo, em algumas situações o teste de Wilcoxon-Mann-Whitney será utilizado com *p-valor*, em inglês *p-value*, igual a 0,05, isto é, com nível de significância de 95%. Outra medida estatística importante e complementar é a medida tamanho de efeito, em inglês *effect-size*. Ela permite identificar quantas vezes um método foi melhor do que outro em uma comparação par-a-par a partir dos resultados observados e permite entender a diferença prática destes resultados. Isso pois o *p-valor* indica apenas que um método é estatisticamente superior a outro, porém não informa o quanto superior. O método pode ser, por exemplo, 1% melhor ou 2.000% melhor. Pode ser melhor em 100% das comparações ou apenas 51% delas. As medidas de tamanho de efeito apresentam esta informação. 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; DELANEY, 2000). De forma simplificada, o valor do tamanho de efeito varia de 0% a 100%. Ele indica o número de vezes em que o método A foi melhor (no caso, maior) do que o método B. Por exemplo, $\hat{A}_{12} = 50\%$ indica que os dois métodos têm chances iguais de gerar o melhor resultado e $\hat{A}_{12} = 80\%$ indica que o primeiro método será melhor que o segundo em 80% das vezes que for aplicado.

A ferramenta R Statistical System v.2.15.2 (THE. . . , 2015) foi utilizada para realização dos testes estatísticos.

4.4 Proposta de solução para o PCMS

Nesta seção define-se, com base em três estudos experimentais, a versão final da heurística proposta para a resolução do problema PCMS baseada na metaheurística BRKGA.

4.4.1 Estudo dos parâmetros do BRKGA

O primeiro estudo experimental (E1) investiga o comportamento do algoritmo BRKGA_PCMS variando-se os valores dos parâmetros do *framework* BRKGA, relacionados na Tabela 5. O objetivo deste estudo é avaliar a influência das configurações do BRKGA considerando-se um único codificador/decodificador. Para tal, selecionou-se a versão básica ($p_v = 0$) do codificador CA e o decodificador BF, apresentados nas Seções 3.1 e 3.2. O racional é que, no processo de atribuição de *cluster* aos módulos, a regra de seleção de *cluster* utilizada pelo decodificador BF escolhe aquele que maximiza o MQ. Uma vez que o objetivo do PCMS também é o de maximizar o MQ, intuitivamente esta regra pareceu ser uma boa escolha para este primeiro estudo. Para este primeiro estudo, utilizou-se o subconjunto de 12 instâncias definidas na Seção 4.2.

Tabela 5: Parâmetros do BRKGA.

Descrição	Parâmetro	Lista de valores
Tamanho da população	p	$20 \times \log_2 n$
Percentual de cromossomos elite	p_e	$(0{,}1; 0{,}15; 0{,}2; 0{,}25)$
Percentual de cromossomos mutantes	p_m	$(0{,}05; 0{,}1; 0{,}15; 0{,}2)$
Percentual de vício de chave aleatória	v_e	$\{0{,}6; 0{,}7; 0{,}8\}$
Número máximo de iterações sem melhoria	max_it	30

Foram avaliadas 13 configurações relacionadas na Tabela 6. A primeira coluna da tabela exibe a identificação da configuração. Para as 12 primeiras configurações (E1C1 até E1C12) adotou-se a estratégia Combinado(p_e/p_m) que indica que os valores de p_e e p_m serão combinados.

Para estas 12 configurações, o algoritmo BRKGA_PCMS possui uma variável atuando como um contador, inicializada com zero, que é incrementada em uma unidade a cada iteração em que não houve melhoria no valor da melhor solução encontrada. Quando o valor desta variável ultrapassar o valor do parâmetro chamado max_it, o valor do contador será atualizado com zero e o algoritmo selecionará novos valores para os parâmetros p_e e p_m, conforme explicado a seguir.

A opção "$0{,}10 \uparrow$" para o parâmetro p_e indica que, da lista de valores relacionados na Tabela 5 para p_e, inicialmente p_e assume o menor deles $(0{,}10)$ e, a cada max_it sem melhoria, muda este valor para o próximo da lista, em ordem crescente. Ao chegar no fim da lista, p_e assume, novamente, o menor valor, e assim sucessivamente.

Analogamente, a opção "$0{,}25\downarrow$" para o parâmetro p_e indica que, da lista de valores relacionados na Tabela 5 para p_e, inicialmente p_e assume o maior deles $(0{,}25)$ e, a cada max_it sem melhoria, muda este valor para o próximo da lista, em ordem decrescente. Ao chegar no início da lista (menor valor), p_e assume, novamente, o maior valor e assim sucessivamente. O mesmo raciocínio é aplicado para o parâmetro p_m.

Tabela 6: Configurações do primeiro estudo experimental.

Configuração	Combinado (p_e/p_m)		v_e
E1C1	$p_e = 0,10\uparrow$ / $p_m = 0{,}20\downarrow$		0,6
E1C2	$p_e = 0,25\downarrow$ / $p_m = 0{,}05\uparrow$		0,6
E1C3	$p_e = 0,10\uparrow$ / $p_m = 0{,}05\uparrow$		0,6
E1C4	$p_e = 0,25\downarrow$ / $p_m = 0{,}20\downarrow$		0,6
E1C5	$p_e = 0,10\uparrow$ / $p_m = 0{,}20\downarrow$		0,7
E1C6	$p_e = 0,25\downarrow$ / $p_m = 0{,}05\uparrow$		0,7
E1C7	$p_e = 0,10\uparrow$ / $p_m = 0{,}05\uparrow$		0,7
E1C8	$p_e = 0,25\downarrow$ / $p_m = 0{,}20\downarrow$		0,7
E1C9	$p_e = 0,10\uparrow$ / $p_m = 0{,}20\downarrow$		0,8
E1C10	$p_e = 0,25\downarrow$ / $p_m = 0{,}05\uparrow$		0,8
E1C11	$p_e = 0,10\uparrow$ / $p_m = 0{,}05\uparrow$		0,8
E1C12	$p_e = 0,25\downarrow$ / $p_m = 0{,}20\downarrow$		0,8
Configuração	p_e	p_m	v_e
E1C13	0,20	0,15	0,7

A configuração E1C13 assume valores fixos para p_e, p_m e v_e.

Em todas as configurações o tamanho da população p é igual a $20 \times \log_2 n$, onde n é a quantidade de módulos e a quantidade de gerações é igual a 1.000. Para cada uma das 12 instâncias, e para cada uma das 13 configurações, foram realizadas 20 execuções independentes, cada uma com uma diferente semente para o gerador de números pseudo-aleatórios, totalizando 3.120 execuções.

Na Tabela 7 são apresentadas, por instância, a melhor média do valor da solução (MQ) de 20 execuções das 13 configurações analisadas e o valor máximo de MQ encontrado em todas as execuções do algoritmo.

Tabela 7: Melhor média e valor máximo do MQ de 13 configurações do BRKGA_PCMS utilizando o codificador CA com $p_v = 0$ e o decodificador BF.

Instância	Melhor Média (MQ)	MQ Máximo
compiler	1,46480	1,46480
xtell	2,00522	2,00522
telnet	1,84749	1,84750
apache	5,67822	5,73670
jscatterplot	10,60007	10,73350
junit	10,70911	10,85070
xntp	7,53987	7,75860
jdendogram	25,51678	25,80380
pdt_renderer	21,51516	21,75410
jung_vizualization	20,72721	21,23450
pfcda_swing	27,21513	27,94720
jml	15,60380	16,24150

Para cada uma das 12 instâncias, a Tabela 8 mostra a média e o desvio padrão do valor de MQ para cada configuração da Tabela 6. Os valores são apresentados em negrito quando atingem as melhores médias obtidas entre as 13 configurações. Além disso, para cada configuração, a penúltima linha da tabela ("# de vitórias") mostra a quantidade de instâncias que obteve a maior média do MQ e a última linha ("# vitórias isoladas") mostra a quantidade de instâncias em que a respectiva configuração foi a única a obter a melhor média de MQ. Nota-se que a configuração que mais vezes atingiu a maior média foi a configuração E1C2, que a obteve em cinco instâncias. Considerando-se o número de vitórias isoladas, as duas melhores configurações são E1C2 e E1C4, ambas com três vitórias.

4.4.1.1 Análise dos resultados

No experimento E1 a configuração E1C13 é a única que possui valores fixos para os parâmetros percentual de mutantes (p_m) e percentual de elite (p_e) (ver Tabela 6), diferentemente da estratégia "Combinado" utilizada em todas as outras 12 configurações. De acordo com os resultados da Tabela 8, é possível verificar que a configuração E1C13 somente obteve a melhor média de MQ nas instâncias "telnetd" e "compiler". Observa-se também que, para a instância "compiler", todas as outras configurações também obtiveram a melhor média de MQ e que para a instância "telnetd" a configuração E1C5 também obteve o melhor resultado.

Sendo assim, é possível concluir que, considerando a amostra selecionada de 12 instâncias teste, a estratégia "Combinado", utilizada nas 12 primeiras configurações, para os parâmetros elite/mutante foi mais eficaz na escolha dos valores dos parâmetros do BRKGA do que a estratégia que usa valores fixos, utilizada na estratégia E1C13 (a estratégia "Combinado" obteve os melhores resultados em 11 das 12 instâncias).

Em relação às outras configurações, E1C1 a E1C12, a análise dos resultados levará em conta as configurações que obtiveram as melhores médias de MQ em pelo menos três instâncias. São elas: E1C1, E1C2, E1C3 e E1C4. A Tabela 9 destaca um resumo com as configurações que atingiram o maior número de vezes as melhores médias em três ou mais instâncias. A coluna "Configuração" mostra o nome da configuração, a segunda e terceira colunas indicam, respectivamente, os valores do percentual de elite (p_e) e do percentual de mutantes (p_m) da opção combinado. A quarta coluna representa o percentual do vício da chave aleatória (v_e) e a última coluna informa o número de vezes em que a respectiva configuração atingiu a melhor média, ou seja, se esta coluna possui um valor igual a 5, significa que a configuração atingiu a melhor média de MQ em cinco instâncias distintas.

É importante destacar a configuração E1C2 que obteve a melhor média de MQ em cinco instâncias incluindo todos os tamanhos definidos (pequeno, médio e grande) e incluindo todas as instâncias ponderadas utilizadas no experimento E1: "compiler" (pequena, não ponderada com 13 módulos), "jml" (a maior do experimento com 267 módulos e não ponderada), "pdfrenderer" (média, não ponderada com 199 módulos), "xtell" (pequena, ponderada com 22 módulos) e

Tabela 8: Média e desvio padrão por instância das 13 configurações analisadas para o BRKGA_PCMS utilizando o codificador CA com $p_v = 0$ e o decodificador BF.

Instância	E1C1	E1C2	E1C3	E1C4	E1C5	E1C6	E1C7	E1C8	E1C9	E1C10	E1C11	E1C12	E1C13
apache	5,66686 ±0, 02781	5,65934 ±0, 04184	5,65898 ±0, 04239	**5,67822 ±0,03942**	5,64706 ±0, 04609	5,65843 ±0, 04424	5,65534 ±0, 04376	5,67112 ±0, 0435	5,66004 ±0, 04291	5,6644 ±0, 04179	5,65684 ±0, 04173	5,659 ±0, 04217	5,64706 ±0, 0444
compiler	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**	**1,46483 ±0**
jdendogram	25,47113 ±0, 15649	25,47793 ±0, 16068	**25,51678 ±0,16079**	25,51409 ±0, 16409	25,39918 ±0, 17709	25,37799 ±0, 20444	25,3627 ±0, 20589	25,50656 ±0, 20093	25,32627 ±0, 21192	25,26362 ±0, 22811	25,28118 ±0, 23665	25,33343 ±0, 23543	25,39918 ±0, 23315
jml	15,45019 ±0, 31941	**15,6038 ±0,35202**	15,45841 ±0, 37336	15,57059 ±0, 35994	15,1555 ±0, 40499	15,10334 ±0, 44144	15,13572 ±0, 46627	15,2834 ±0, 45583	14,96664 ±0, 48721	14,73517 ±0, 52093	14,66868 ±0, 55349	14,8065 ±0, 57161	15,1555 ±0, 56253
jscatterplot	10,58035 ±0, 06572	10,59527 ±0, 06354	**10,60007 ±0,06514**	10,59484 ±0, 0592	10,5842 ±0, 06257	10,58169 ±0, 06917	10,58218 ±0, 07059	10,59609 ±0, 07007	10,55487 ±0, 07373	10,56599 ±0, 07767	10,53829 ±0, 08238	10,57792 ±0, 08218	10,5842 ±0, 08159
jung_vizualization	**20,72721 ±0,21778**	20,69376 ±0, 29739	20,71591 ±0, 2639	20,66554 ±0, 2771	20,48251 ±0, 31814	20,51442 ±0, 31749	20,50027 ±0, 32846	20,64719 ±0, 32134	20,24344 ±0, 38186	20,32051 ±0, 38697	20,45594 ±0, 37959	20,38011 ±0, 38141	20,48251 ±0, 38304
junit	10,70784 ±0, 08883	10,70495 ±0, 08497	10,70722 ±0, 08371	**10,70911 ±0,08996**	10,68235 ±0, 09941	10,61255 ±0, 10834	10,59365 ±0, 1232	10,65016 ±0, 12242	10,61006 ±0, 12716	10,59042 ±0, 13673	10,62429 ±0, 13875	10,56091 ±0, 14812	10,68235 ±0, 147
pdf_renderer	21,43908 ±0, 18381	**21,51516 ±0,16585**	21,44902 ±0, 14723	21,48719 ±0, 15126	21,41632 ±0, 1599	21,4092 ±0, 1563	21,43255 ±0, 15636	21,46497 ±0, 15443	21,35644 ±0, 15567	21,31156 ±0, 17001	21,33495 ±0, 17145	21,3125 ±0, 17981	21,41632 ±0, 1801
pfcda_swing	27,12685 ±0, 44534	27,09749 ±0, 41825	27,17972 ±0, 40229	**27,21513 ±0,39589**	26,8984 ±0, 42622	27,04642 ±0, 45624	26,78427 ±0, 48529	26,86321 ±0, 4896	26,25283 ±0, 55204	26,32794 ±0, 60049	26,68932 ±0, 59726	26,59816 ±0, 59813	26,8984 ±0, 58987
telnetd	1,84724 ±0, 00114	1,84724 ±0, 00113	1,84699 ±0, 00128	1,84724 ±0, 00124	**1,84749 ±0,00112**	1,84724 ±0, 00112	1,84673 ±0, 00126	1,84673 ±0, 00135	1,84699 ±0, 00137	1,84699 ±0, 00139	1,84623 ±0, 0015	1,84712 ±0, 00148	**1,84749 ±0,00143**
xntp	7,52447 ±0, 04138	**7,53987 ±0,04182**	7,51822 ±0, 0402	7,51781 ±0, 04325	7,49546 ±0, 05624	7,5356 ±0, 05748	7,53694 ±0, 05739	7,52805 ±0, 06102	7,51467 ±0, 06247	7,48948 ±0, 0641	7,51225 ±0, 06495	7,51243 ±0, 0637	7,49546 ±0, 06609
xtell	**2,00522 ±3E-05**	**2,00522 ±3E-05**	2,00519 $\pm 4E-05$	2,00519 $\pm 4E-05$	2,00521 $\pm 4E-05$	2,00518 $\pm 4E-05$	2,00521 $\pm 4E-05$	2,00518 $\pm 4E-05$	2,00518 $\pm 4E-05$	2,00516 $\pm 4E-05$	2,00517 $\pm 4E-05$	2,00516 $\pm 5E-05$	2,00521 $\pm 4E-05$
# de vitórias	3	5	3	4	2	1	1	1	1	1	1	1	2
# vitórias isoladas	1	3	2	3	0	0	0	0	0	0	0	0	0

Tabela 9: Resumo das configurações que atingiram a melhor média em três ou mais instâncias.

Configuração	p_e	p_m	v_e	Vitórias
E1C2	0,25 ↓	0,05 ↑	0,6	5
E1C4	0,25 ↓	0,20 ↓	0,6	4
E1C1	0,10 ↑	0,20 ↓	0,6	3
E1C3	0,10 ↑	0,05 ↑	0,6	3

"xntp" (média, ponderada com 111 módulos).

A configuração E1C4 obteve êxito em quatro instâncias: "apache" (pequena, não ponderada com 36 módulos), "compiler" (pequena, não ponderada com 13 módulos), "junit" (média, não ponderada com 100 módulos) e "pfcda_swing" (a segunda maior instância com 252 módulos e não ponderada).

Em relação aos parâmetros utilizados nestas quatro configurações é importante destacar a probabilidade do vício da chave aleatória que em todas configurações assumiu o valor de 60%. De acordo com estes resultados, novamente considerando a amostra selecionada de 12 instâncias teste, este é o parâmetro de vício mais eficaz do BRKGA_PCMS utilizando o codificador CA com $p_v = 0$ e o decodificador BF.

As quatro configurações com melhores médias foram comparadas aplicando-se o teste não paramétrico Wilcoxon-Mann-Whitney. A Tabela 10 apresenta os resultados comparativos dos testes estatísticos entre as quatro configurações que obtiveram as melhores médias de MQ (E1C1, E1C2, E1C3 e E1C4). A primeira coluna da tabela exibe o nome da instância. Em seguida, para cada uma das seis combinações de pares possíveis entre as quatro configurações, a tabela exibe o *p-valor* (p) e o tamanho de efeito (es). Os *p-valores* 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.

Tabela 10: *P-valor* e tamanho de efeito na comparação entre as quatro melhores configurações do BRKGA_PCMS.

Instância	E1C1 > E1C2		E1C1 > E1C3		E1C1 > E1C4		E1C2 > E1C3		E1C2 > E1C4		E1C3 > E1C4	
	p	es	p	es	p	es	p	es	p	es	p	es
compiler	'-'	50%	'-'	50%	'-'	50%	'-'	50%	'-'	50%	'-'	50%
xtell	1	50%	0,7516	**52%**	0,6728	50%	0,7516	**52%**	0.6728	50%	0.9198	48%
telnet	1	50%	0,9174	50%	1	49%	0,9174	50%	1	50%	0,9174	49%
apache	0,7512	49%	0,8229	49%	0,6746	**51%**	0,9298	47%	0,5522	49%	0,5905	**51%**
jscatterplot	0,7851	48%	0,6286	47%	0,7187	49%	0,9763	49%	0,9877	50%	0,8611	**51%**
junit	0,7694	**51%**	0,6398	**51%**	0,9549	**51%**	0,8733	49%	0,7931	49%	0,677	49%
xntp	0,662	47%	0,6997	47%	0,6845	48%	0,8855	50%	0,9426	50%	0,9631	50%
jdendogram	0,8977	49%	0,7971	47%	0,527	**52%**	0,9385	48%	0,6471	**52%**	0,7931	**54%**
pdfrenderer	0,6342	47%	0,947	49%	0,882	**51%**	0,481	**52%**	0,825	**54%**	0,662	**51%**
jung_vizualization	0,9437	50%	0,8259	50%	0,7315	**54%**	0,7418	49%	0,8727	**52%**	0,56	**53%**
pfcda_swing	0,9178	40%	0,8249	50%	0,6364	**56%**	0,8752	50%	0,7173	**56%**	0,5549	**56%**
jml	0,7692	48%	0,7799	49%	1	**55%**	0,9763	50%	0,8849	**56%**	0,8139	**56%**

Um *p-valor* menor do que 0,05 (em negrito) indica resultados significativamente diferentes com 95% de certeza. Os dados da Tabela 10 mostram que em nenhuma das comparações realizadas os resultados obtidos são estatisticamente diferentes. Consequentemente não é possí-

vel afirmar, com os dados disponíveis, que uma das quatro configurações seja estatisticamente diferente das outras três.

Em relação ao tamanho de efeito, a comparação entre as configurações E1C1 e E1C2 (configuração com maior número de vitórias) apresentou um tamanho de efeito favorável para a configuração E1C2 em sete instâncias sendo que para a segunda maior instância, a "pfcda_swing", obteve um efeito favorável de 60%.

Comparando-se as configurações E1C1 e E1C3, percebe-se um tamanho de efeito ligeiramente superior para E1C1 em três das quatro instâncias médias, na maior instância de todas ("jml") e em uma das instâncias pequenas.

A comparação entre as configurações E1C1 e E1C4 (segunda melhor em número de vitórias) mostra que a configuração E1C1 tem um tamanho de efeito favorável em todas as instâncias grandes e em três das quatro instâncias médias.

Os resultados das configurações E1C2 e E1C3 empatam em cinco das 12 instâncias. Entretanto, E1C3 apresenta um tamanho de efeito favorável em pelo menos uma das instâncias de todos os tamanhos.

A comparação entre as duas configurações com o maior número de vitórias (E1C2 e E1C4) destaca um tamanho de efeito favorável para E1C2 em todas as instâncias grandes e em duas das maiores instâncias médias. A configuração E1C4 somente obteve um tamanho de efeito favorável em uma instância pequena e uma média, ambas com 51%.

Por último, a comparação entre as configurações E1C3 e E1C4 mostra um tamanho de efeito favorável para E1C3 em todas as instâncias grandes e em duas das maiores instâncias médias além de uma leve superioridade na instância "apache" (pequena) e na "jscatterplot" (menor das instâncias médias).

Para resumir, tal resultado mostra que: (i) o parâmetro v_e parece ter influenciado a qualidade das soluções, pois as quatro configurações com maior quantidade de vitórias possuem $v_e = 0{,}6$; (ii) para os parâmetros p_e e p_m a estratégia combinada se mostrou melhor do que a que usa valores fixos; e (iii) entre as estratégias que usam valores combinados e valores crescente/descrente para os parâmetros p_e e p_m, todas parecem equivalentes. Apesar de nenhum dos resultados ser estatisticamente significante com grau de confiança de 95%, os tamanhos de efeito na comparação das duas configurações com maior número de vitórias mostram superioridade da configuração E1C2. Sendo assim, para o próximo estudo computacional, entre as 13 configurações será selecionada aquela que obteve a maior quantidade de vitórias, a configuração E1C2.

4.4.2 Estudo dos decodificadores

O segundo estudo experimental (E2) investiga o comportamento do algoritmo variando-se os decodificadores, conforme apresentado nas Seções 3.2 e 3.4. Para tal, considerou-se a melhor versão do BRKGA obtida no estudo anterior, no caso a configuração E1C2.

Foram avaliadas 24 configurações relacionadas na Tabela 11. A primeira coluna da tabela exibe a identificação da configuração. As duas próximas colunas exibem, respectivamente, o identificador do codificador e do decodificador, conforme apresentado no Capítulo 3. A quarta coluna exibe o valor do parâmetro percentual de vizinhos (p_v).

Todas as configurações usam os valores de parâmetros usados em E1C2 (Tabela 6), que são: $p_e = 0{,}25 \downarrow$, $p_m = 0{,}05 \uparrow$ e $v_e = 0{,}6$. Além disso, em todas as configurações o tamanho da população p é igual a $20 \times \log_2 n$, onde n é a quantidade de módulos, e a quantidade de gerações é igual a 1.000. Para cada uma das configurações foram realizadas 20 execuções independentes, cada uma com uma diferente semente para o gerador de números pseudo-aleatórios, totalizando 5.760 execuções.

Tabela 11: Configurações do segundo estudo computacional (E2). Todas as configurações usam $p_e = 0{,}25 \downarrow$, $p_m = 0{,}05 \uparrow$ e $v_e = 0{,}6$, com população $p = 20 \times \log_2 n$.

Configuração	Codificador	Decodificador	p_v
E2C1	CA	FF	0
E2C2	CA	LBF	0
E2C3	CA	WF	0
E2C4	CB	LBF+BF	1
E2C5	CA	BF	0,2
E2C6	CA	FF	0,2
E2C7	CA	LBF	0,2
E2C8	CA	WF	0,2
E2C9	CB	LBF+BF	0,7
E2C10	CA	BF	0,5
E2C11	CA	FF	0,5
E2C12	CA	LBF	0,5
E2C13	CA	WF	0,5
E2C14	CB	LBF+BF	0,5
E2C15	CA	BF	0,7
E2C16	CA	FF	0,7
E2C17	CA	LBF	0,2
E2C18	CA	WF	0,7
E2C19	CB	LBF+BF	0,2
E2C20	CA	BF	1
E2C21	CA	FF	1
E2C22	CA	LBF	1
E2C23	CA	WF	1
E2C24	CB	LBF+BF	0

Na Tabela 12 são apresentadas, por instância, a melhor média do valor da solução (MQ) de 20 execuções entre as 24 configurações analisadas e o valor máximo de MQ encontrado em todas as execuções do algoritmo.

Tabela 12: Melhor média e valor máximo do MQ das 24 configurações BRKGA_PCMS usando o codificador CA e CB os decodificadores BF, FF, LBF, WF e BF+LBF.

Instância	Melhor Média (MQ)	MQ Máximo
compiler	1,50505	1,50650
xtell	2,00523	2,00523
telnet	1,84749	1,84750
apache	5,762652	5,76640
jscatterplot	10,70764	10,7419
junit	10,86665	10,9835
xntp	8,07828	8,2088
jdendogram	25,66640	25,9135
pdf_renderer	22,05934	22,17300
jung_vizualization	21,01626	21,4346
pfcda_swing	28,11342	28,4651
jml	16,16852	16,5297

Tabela 13: Média e desvio padrão por instância das quatro configurações analisadas para o BRKGA_PCMS utilizando o codificador CA e o decodificador BF.

Instância	E2C5	E2C10	E1C15	E2C20
apache	5,64141 $\pm 0,02758$	5,58242 $\pm 0,05851$	5,70993 $\pm 0,07347$	**5,76265** **$\pm$0,08197**
compiler	1,23156 ± 0	1,42036 $\pm 0,0956$	1,34643 $\pm 0,07833$	1,49683 $\pm 0,09843$
jdendogram	24,94196 $\pm 0,26109$	24,40089 $\pm 0,36317$	25,34354 $\pm 0,45552$	**25,6664** **$\pm$0,52111**
jml	14,52512 $\pm 0,4343$	15,02067 $\pm 0,43008$	15,79071 $\pm 0,60421$	16,11128 $\pm 0,68412$
jscatterplot	10,41666 $\pm 0,08011$	10,50885 $\pm 0,08338$	**10,70764** **$\pm$0,13816**	10,61187 $\pm 0,12619$
jung_vizualization	20,9408 $\pm 0,17572$	20,59645 $\pm 0,23158$	20,73507 $\pm 0,20796$	**21,01626** **$\pm$0,22412**
junit	10,03927 $\pm 0,11786$	9,88857 $\pm 0,141$	10,30167 $\pm 0,2008$	10,84742 $\pm 0,38124$
pdf_renderer	20,76585 $\pm 0,31059$	20,7051 $\pm 0,26942$	21,32799 $\pm 0,36371$	21,90813 $\pm 0,52991$
pfcda_swing	26,56181 $\pm 0,42347$	27,37464 $\pm 0,55312$	27,91565 $\pm 0,64404$	**28,11342** **$\pm$0,67308**
telnetd	1,83205 $\pm 0,00881$	1,70372 $\pm 0,06604$	1,70864 $\pm 0,06177$	1,40529 $\pm 0,15876$
xntp	7,23769 $\pm 0,12874$	7,19009 $\pm 0,11844$	7,24064 $\pm 0,12564$	7,13684 $\pm 0,13574$
xtell	**2,00523** **$\pm$0**	1,88396 $\pm 0,07974$	1,71433 $\pm 0,12723$	1,68837 $\pm 0,13491$
Número de vitórias	1	0	1	4

4.4.2.1 Análise dos resultados

Para cada uma das 12 instâncias, as Tabelas 13, 14, 15 e 16 mostram a média e o desvio padrão do valor de MQ para cada configuração da Tabela 11. Os valores são apresentados em negrito quando atingem as melhores médias obtidas entre as 24 configurações. Além disso, a última linha de cada tabela mostra a quantidade de vezes (número de instâncias) que cada

Tabela 14: Média e desvio padrão por instância das cinco configurações analisadas para o BRKGA_PCMS utilizando o codificador CB e o decodificador LBF+BF.

Instância	E2C24	E2C19	E2C14	E2C9	E2C4
apache	5,71011 ±0, 04962	5,69793 ±0, 05203	5,61715 ±0, 07201	5,72696 ±0, 06974	5,73698 ±0, 06662
compiler	**1,50505 ±0,00217**	1,5036 ±0, 00373	1,50408 ±0, 00391	1,40715 ±0, 04262	1,50505 ±0, 03939
jdendogram	25,28606 ±0, 30284	24,94475 ±0, 34685	24,4539 ±0, 44617	25,39801 ±0, 457	25,48678 ±0, 45391
jml	15,82598 ±0, 48621	14,52202 ±0, 79194	15,04717 ±0, 66089	15,78368 ±0, 65192	**16,16852 ±0,68699**
jscatterplot	10,59618 ±0, 12214	10,37436 ±0, 1935	10,49029 ±0, 16573	10,68454 ±0, 17078	10,60122 ±0, 15826
jung_vizualization	20,91988 ±0, 37566	20,91848 ±0, 28953	20,8687 ±0, 24927	20,91847 ±0, 23097	20,96095 ±0, 22069
junit	10,83644 ±0, 0892	10,46399 ±0, 21832	10,29672 ±0, 26029	10,6152 ±0, 23319	**10,86665 ±0,24793**
pdf_renderer	**22,05934 ±0,15871**	21,37969 ±0, 38792	20,84272 ±0, 55602	21,68636 ±0, 49767	22,01978 ±0, 49392
pfcda_swing	27,53553 ±0, 29411	26,86641 ±0, 47197	27,37834 ±0, 42755	27,83625 ±0, 47007	28,09407 ±0, 51141
telnetd	**1,84749 ±0**	**1,84749 ±0**	1,84165 ±0, 00922	1,84325 ±0, 00896	1,8437 ±0, 00988
xntp	**8,07828 ±0,08217**	7,38872 ±0, 36368	7,34815 ±0, 36192	7,33925 ±0, 34012	7,13883 ±0, 34893
xtell	**2,00523 ±0**	1,99658 ±0, 01389	1,94352 ±0, 06164	1,9646 ±0, 06087	1,98867 ±0, 05614
Número de vitórias	**5**	1	0	0	2

configuração obteve a maior média do MQ.

No experimento E2 a configuração E2C24 (LBF+BF com 0% dos vizinhos e valores dos demais parâmetros iguais aos de E1C2) atingiu a melhor média de MQ em cinco instâncias: todas as instâncias ponderadas (“xntp” de tamanho Médio com 111 módulos e “xtell” de tamanho pequeno com 22 módulos); uma instância pequena(“telnetd” com 28 módulos) e uma instância média (“pdf_renderer” com 199 módulos).

A configuração E2C20, da Tabela 13 (idêntica a configuração E1C2 só que com 100% dos vizinhos), obteve a melhor média de MQ em 4 instâncias: uma pequena (“apache” com 36 módulos); uma média (“jdendogram” com 177 módulos) e duas instâncias grandes (“pfcda_swing” com 252 módulos e “jung_vizualization” com 221 módulos).

É perceptível também que os codificadores WF e LBF isoladamente (Tabela 16) apresentaram os piores resultados dentre todos os codificadores: não obtiveram nenhuma vitória nas suas cinco configurações.

Em outra análise possível, nota-se que o Método LBF+BF, que escolhe a partir de uma probabilidade de 50% a estratégia LBF ou a estratégia BF, é mais eficiente que as estratégias que trabalham apenas com um decodificador. Percebe-se que em oito instâncias este método

Tabela 15: Média e desvio padrão por instância das cinco configurações analisadas para o BRKGA_PCMS utilizando o codificador CA e o decodificador FF.

Instância	E2C1	E2C6	E1C11	E2C16	E2C21
apache	5,6464 $\pm 0,12248$	5,59571 $\pm 1,31479$	5,53339 $\pm 1,21995$	5,71825 $\pm 1,14223$	5,75284 $\pm 1,06857$
compiler	1,46367 ± 0	1,40527 $\pm 0,2517$	1,39842 $\pm 0,23467$	1,32143 $\pm 0,20608$	1,49683 $\pm 0,20755$
jdendogram	24,53908 $\pm 0,45182$	23,64405 $\pm 9,8483$	22,71712 $\pm 9,03202$	23,60344 $\pm 8,34122$	25,20335 $\pm 7,91771$
jml	13,98605 $\pm 0,2505$	13,45001 $\pm 5,16622$	14,38117 $\pm 5,08497$	15,30292 $\pm 4,89428$	15,96665 $\pm 4,71196$
jscatterplot	10,41688 $\pm 0,29374$	10,20691 $\pm 3,36014$	10,11564 $\pm 3,13151$	10,25559 $\pm 2,88424$	10,50469 $\pm 2,69467$
jung_vizualization	19,55874 $\pm 0,47648$	19,82572 $\pm 8,01578$	19,78031 $\pm 7,51322$	20,20205 $\pm 6,94726$	20,89648 $\pm 6,51634$
junit	10,67813 $\pm 0,33078$	10,29388 $\pm 3,29641$	10,06612 $\pm 3,04227$	10,32334 $\pm 2,80984$	10,80326 $\pm 2,65959$
pdf_renderer	21,28165 $\pm 0,44151$	20,92059 $\pm 8,5163$	20,08318 $\pm 7,8067$	20,92208 $\pm 7,21719$	21,77401 $\pm 6,78543$
pfcda_swing	24,06257 $\pm 0,48688$	24,2844 $\pm 10,24382$	27,00953 $\pm 10,32256$	27,35431 $\pm 9,72724$	27,992 $\pm 9,17287$
telnetd	1,71296 $\pm 0,04349$	1,75018 $\pm 0,06387$	1,64604 $\pm 0,06038$	1,62025 $\pm 0,06102$	1,41786 $\pm 0,11503$
xntp	7,05382 $\pm 0,24711$	6,73715 $\pm 1,58208$	6,82797 $\pm 1,50475$	6,99644 $\pm 1,4147$	7,07703 $\pm 1,33147$
xtell	1,93296 $\pm 0,01986$	1,78948 $\pm 0,11773$	1,71584 $\pm 0,09776$	1,69386 $\pm 0,08461$	1,49507 $\pm 0,11001$
Número de vitórias	0	0	0	0	0

conseguiu obter as melhores médias de MQ apenas com cinco configurações (E2C4, E2C9, E2C14, E2C19, E2C24) enquanto que as outras 19 configurações, que utilizam os decodificadores simples (BF, FF, LBF e WF), conseguiram obter as melhores médias apenas em seis instâncias.

Na Tabela 17 é apresentado um resumo das configurações que atingiram a melhor média em quatro ou mais instâncias. Estas duas configurações com melhores médias (E2C24 e E2C20) foram comparadas aplicando-se o teste não paramétrico Wilcoxon-Mann-Whitney com um nível de significância de 95%, além da medida de tamanho de efeito, que permite identificar quantas vezes um método foi melhor do que o outro em uma comparação par-a-par. A Tabela 18 apresenta os resultados comparativos destes testes estatísticos. A primeira coluna da tabela exibe o nome da instância. Em seguida, para a combinação destas duas configurações, a tabela exibe o *p-valor* (p) e o tamanho de efeito (es). Os valores de tamanho de efeito maiores do que 50% são apresentados em negrito, assim como *p-valores* menores do que 5%.

Em relação ao tamanho de efeito, a comparação entre as configurações E2C24 e E2C20 apresentou um tamanho de efeito favorável para a configuração E2C20 em seis instâncias (inclusive as três maiores instâncias "jml", "pfcda_swing" e "jung_vizualization") apresentando uma média de 54,8%. A configuração E2C24, apesar de ter tido um tamanho de efeito favorável

Tabela 16: Média e desvio padrão por instância das dez configurações analisadas para o BRKGA_PCMS utilizando o codificador CA os decodificadores LBF e WF.

Instância	E2C3	E2C2	E2C7	E2C8	E2C12	E1C13	E2C17	E2C18	E2C22	E2C23
apache	3,00789 $\pm 0,05651$	5,35542 $\pm 0,17018$	5,35576 $\pm 0,16585$	3,22985 $\pm 0,98282$	5,52337 $\pm 0,91412$	4,63042 $\pm 0,84694$	5,72138 $\pm 0,82858$	5,25173 $\pm 0,77718$	5,7098 $\pm 0,75825$	5,658 $\pm 0,73474$
compiler	0,90833 $\pm 0,00519$	1,33679 $\pm 0,06455$	1,375 $\pm 0,05383$	0,90833 $\pm 0,21576$	1,36457 $\pm 0,197$	1,10476 $\pm 0,19254$	1,34547 $\pm 0,18073$	1,17419 $\pm 0,17202$	1,46483 $\pm 0,17461$	1,46483 $\pm 0,17449$
jdendogram	4,21541 $\pm 0,39451$	19,93071 $\pm 2,37521$	20,84371 $\pm 2,05513$	8,50635 $\pm 6,04946$	21,94266 $\pm 5,58383$	16,26591 $\pm 5,2075$	23,00326 $\pm 5,05317$	20,50023 $\pm 4,74218$	24,89069 $\pm 4,78823$	23,91062 $\pm 4,68792$
jml	3,26779 $\pm 0,67298$	12,20978 $\pm 1,13761$	12,4512 $\pm 1,02144$	6,94879 $\pm 2,73638$	14,03833 $\pm 2,66816$	11,81436 $\pm 2,43604$	15,04893 $\pm 2,51038$	13,9642 $\pm 2,40943$	15,94467 $\pm 2,50959$	15,84281 $\pm 2,5366$
jscatterplot	3,5937 $\pm 0,21998$	8,40335 $\pm 1,04217$	8,42698 $\pm 0,9806$	4,50024 $\pm 2,17333$	9,6592 $\pm 2,06251$	8,6165 $\pm 1,88651$	9,99197 $\pm 1,84087$	9,7323 $\pm 1,76506$	10,478 $\pm 1,75368$	10,38293 $\pm 1,72109$
jung_vizualization	4,01776 $\pm 0,55555$	15,75914 $\pm 1,98542$	17,71055 $\pm 1,62399$	9,26454 $\pm 3,92785$	19,4316 $\pm 3,83764$	14,43033 $\pm 3,5746$	20,10438 $\pm 3,60489$	18,08405 $\pm 3,4067$	20,80816 $\pm 3,45229$	19,95065 $\pm 3,37536$
junit	3,80202 $\pm 0,12272$	10,05117 $\pm 0,3665$	10,00672 $\pm 0,35922$	5,0612 $\pm 2,28198$	10,18156 $\pm 2,0984$	6,87357 $\pm 2,10346$	10,49421 $\pm 2,03451$	8,7685 $\pm 1,90492$	10,78316 $\pm 1,88063$	10,3239 $\pm 1,81514$
pdf_renderer	4,12135 $\pm 0,22043$	19,62012 $\pm 0,91212$	20,29639 $\pm 0,75959$	8,09157 $\pm 5,40587$	19,92127 $\pm 4,94307$	12,56223 $\pm 4,92459$	20,97085 $\pm 4,7697$	18,7413 $\pm 4,47899$	21,64171 $\pm 4,40173$	21,04402 $\pm 4,26681$
pfcda_swing	4,08862 $\pm 0,95849$	20,05364 $\pm 2,16722$	21,15747 $\pm 1,83277$	11,65667 $\pm 4,68088$	26,07629 $\pm 5,00646$	22,2498 $\pm 4,60953$	27,07082 $\pm 4,78892$	25,63696 $\pm 4,6596$	27,9176 $\pm 4,74202$	27,24754 $\pm 4,68609$
telnetd	1,6629 $\pm 0,0271$	1,44487 $\pm 0,13873$	1,49234 $\pm 0,13332$	1,70542 $\pm 0,13486$	1,56277 $\pm 0,12143$	1,59459 $\pm 0,11091$	1,5105 $\pm 0,11053$	1,45681 $\pm 0,11598$	1,41786 $\pm 0,11813$	1,41103 $\pm 0,11899$
xntp	3,64745 $\pm 0,30862$	5,43178 $\pm 0,85554$	6,39526 $\pm 0,7071$	4,65191 $\pm 0,94472$	6,48132 $\pm 0,88422$	5,62275 $\pm 0,827$	6,85498 $\pm 0,83164$	6,57996 $\pm 0,79768$	6,91041 $\pm 0,7913$	6,7245 $\pm 0,76993$
xtell	1,55861 $\pm 0,06872$	1,58937 $\pm 0,18357$	1,64119 $\pm 0,16014$	1,56163 $\pm 0,15689$	1,70913 $\pm 0,14065$	1,60244 $\pm 0,13297$	1,65477 $\pm 0,12435$	1,55011 $\pm 0,12297$	1,47472 $\pm 0,1291$	1,54092 $\pm 0,1264$
Número de vitórias	0	0	0	0	0	0	0	0	0	0

em uma instância a menos, no caso cinco instâncias, tem uma média de tamanho de efeito maior (estas cinco instâncias têm uma média de 57%) e na instância "telnet" apresentou um tamanho de efeito favorável de 62%.

Em relação ao *p-valor*, percebe-se que os resultados das 24 execuções podem ser considerados estatisticamente distinto com um nível de certeza de 95% em duas das instâncias, "compiler" e "telnet", ambas instâncias pequenas.

Considerando a vantagem da maior média de tamanho de efeito apresentada pela configuração E2C24, decidiu-se escolher esta configuração para a versão final do algoritmo BRKGA_PCMS.

Tabela 17: Resumo das configurações que atingiram a melhor média em quatro ou mais instâncias.

Configuração	p_e	p_m	v_e	p_v	Vitórias
E2C4	0,25 ↓	0,05 ↑	0,6	1	5
E2C20	0,25 ↓	0,05 ↑	0,6	1	4

Tabela 18: *P-valor* e tamanho de efeito na comparação entre as duas melhores configurações do Experimento E2.

Instância	E2C24 > E2C20	
	p	es
compiler	**0,04712**	**61%**
xtell	1	50%
telnet	**0,03795**	**62%**
apache	0.2372	42%
jscatterplot	0,1312	**59%**
junit	0,7537	48%
xntp	0,5463	**53%**
jdendogram	0,1999	42%
pdfrenderer	0,8487	**51%**
jung_vizualization	0,926	49%
pfcda_swing	0,3128	43%
jml	0,7472	47%

4.4.3 Busca local

O segundo estudo experimental, apresentado na seção anterior, elegeu a configuração E2C24 como a melhor configuração do algoritmo BRKGA_PCMS. O terceiro estudo experimental, apresentado nesta seção, avalia a aplicação do procedimento de busca local, apresentado na Seção 3.5, à solução obtida pelo algoritmo BRKGA_PCMS. O algoritmo resultante será denominado de BRKGA_PCMS_BL.

Para este terceiro experimento serão utilizadas as mesmas 12 instâncias do experimento anterior com a configuração E2C24. O objetivo do estudo é avaliar se a eventual melhora

da qualidade da solução obtida com a aplicação da busca local compensa o tempo adicional necessário para executá-la. O critério de parada da parte da heurística anterior à execução da busca local é o seguinte: seja *max_iterações* o número máximo de iterações, *min_iterações* o número mínimo de iterações e *max_iterações_sem_melhoria* o número máximo de iterações sem melhoria. Inicialmente, o algoritmo BRKGA_PCMS executa *min_iterações*. Em seguida, ele pára quando a primeira de uma das duas condições for verdadeira: *max_iterações_sem_melhoria* ou *max_iterações*. Neste estudo os seguintes valores foram utilizados para os parâmetros: *min_iterações*= 1.000, *max_iterações_sem_melhoria* = 500 e *max_iterações* = 3.000. Para cada uma das 12 instâncias, foram realizadas 20 execuções independentes, totalizando 240 execuções.

A primeira coluna da Tabela 19 apresenta a identificação da instância, a segunda e a terceira colunas apresentam a média dos valores de MQ e do tempo de execução da heurística BRKGA_PCMS. Em seguida, a terceira e a quarta colunas apresentam as médias de MQ e de tempo de execução da heurística BRKGA_PCMS_BL. A última coluna "*Gap*(%)" reporta o valor do percentual de melhoria da heurística BRKGA_PCMS_BL calculado como ((BRKGA_PCMS_BL - BRKGA_PCMS)/BRKGA_PCMS_BL)*100.

Analisando os resultados exibidos na Tabela 19 é possível perceber que a busca local melhorou a qualidade da solução em 1,06% (*gap*) com o custo de piorar o tempo total de execução em 3,91 % . Em especial, destaca-se os resultados obtidos para as instâncias "jml" e "pfcda_swing" que tiveram uma melhoria de MQ de 4,45% e 1,73% e uma piora no tempo de menos de 0,001%.

Acredita-se que a melhoria da qualidade de solução de 1,06% obtida pela utilização da busca local nas médias dos valores de MQ compensa o esforço computacional adicional de 3,91% e, portanto, o algorimto BRKGA_PCMS_BL será utilizado nos estudos computacionais realizados na próxima seção.

4.5 Comparação do BRKGA_PCMS_BL com diferentes abordagens da literatura

Nesta seção deseja-se comparar a versão final do algoritmo proposto BRKGA_PCMS_BL com diferentes abordagens da literatura para resolver o problema de clusterização de módulos de software. Inicialmente, os estudos foram divididos em dois grupos em função do tipo de abordagem heurística da literatura e do subconjunto de instâncias teste utilizado nos experimentos. Por fim, o último estudo apresenta um resumo reunindo todas as instâncias abordadas neste trabalho e os melhores resultados obtidos na literatura.

Tabela 19: Comparação do BRKGA_PCMS_BL com o BRKGA_PCMS para 12 instâncias.

	BRKGA_PCMS_BL		BRKGA_PCMS		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)	
apache	**5,74344**	9,40790	5,71011	**9,40745**	0,58
compiler	**1,50505**	1,58460	1,50505	1,58450	0
jdendogram	**25,66695**	309,85555	25,28606	**172,35000**	1,48
jml	**16,56364**	1235,7595	15,82598	1235,73730	4,45
jscatterplot	**10,65829**	54,94035	10,59618	**54,93945**	0,58
jung_visualization	**21,33284**	651,7443	20,91988	651,73320	1,93
junit	**10,88078**	74,13880	10,83644	74,13750	0,40
pdf_renderer	**22,18628**	437,94420	22,05934	**437,93680**	0,57
pfcda_swing	**28,0218**	734,0917	27,53553	734,07795	1,73
telnetd	**1,84749**	5,73535	1,87490	5,73535	0
xntp	**8,16141**	128,42590	8,07828	128,42325	1,01
xtell	**2,00523**	3,82395	2,00523	3,82400	0
Média *gap*					1,06
Tempo total		3647,4521		3509,88675	

4.5.1 Adequação do tempo de processamento

Os experimentos computacionais realizados com as instâncias da literatura foram executados em processadores diferentes daquele utilizado nesta dissertação. Tal fato exige que seja realizada uma adequação do tempo de execução destes experimentos de modo a simular que eles tenham sido executados em uma mesma máquina (QUIROZ-CASTELLANOS et al., 2015). Para atingir este objetivo serão calculados fatores de escala (em inglês, *scale factors*) para cada processador a partir dos *benchmarks* reportados nos testes do SPEC *Standards* (http://www.spec.org/benchmarks.html). Este *site* reporta dados para diferentes *benchmarks*. Por exemplo, os *benchmarks* CINT2006 e CFP2006 reportam a medida de tempo para se executar uma tarefa considerando, respectivamente, variáveis do tipo *int* e *float point*. Neste trabalho considerou-se o resultado reportado na coluna "Result" do *benchmark* CINT2006. Como os processadores deste experimento e os processadores utilizados na literatura possuem, para um mesmo *benchmark*, resultados distintos entre si, eles serão adequados tomando como base a velocidade de CPU (coluna "Result" do *benchmark*) do processador desta dissertação (as velocidades de CPU dos processadores utilizados na literatura serão divididas pela velocidade de CPU do processador utilizado nos experimentos desta dissertação). Por fim, o resultado desta divisão, chamado de fator de escala, será utilizado para multiplicar o tempo computacional obtido pelos métodos da literatura a serem comparados. Os seguintes algoritmos da literatura são utilizados no exeperimento comparativo: o algoritmo *Grouping Number Encoding* (PINTO, 2014), chamado de AG_GNE_LIT; o algoritmo *Grouping Genetic Algorithm* (PINTO, 2014), chamado de AG_GGA_LIT; o algoritmo *Grouping Number Encoding* (PRADITWONG, 2011), chamado de GNE_LIT; o algoritmo *Grouping Genetic Algorithm* (PRADITWONG, 2011), chamado de GGA_LIT; a heurística ILS_CMS (PINTO, 2014) e o método exato $\text{MILP}^{+}_{2,1}$ proposto

por Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013). A Tabela 20 possui estas informações, sendo que em dois casos (GNE_LIT e GGA_LIT) não foi possível identificar o processador utilizado. Nestes casos, considera-se que os tempos computacionais destes dois métodos não são comparáveis com o tempo de BRKGA_PCMS_BL.

Tabela 20: Resultado do *benchmark* CINT2006 e fatores de escala usados para comparar o desempenho de métodos da literatura que trataram o problema de PCMS.

Método	Processador	Velocidade da CPU	Fator de escala
BRKGA_PCMS_BL	XEON CPU E5-1607	43,1	43,1/43,1 = 1,00
GNE_LIT	Dados não disponíveis	-	-
GGA_LIT	Dados não disponíveis	-	-
AG_GNE_LIT	Intel Core i7-2600	45,1	43,1/45,1 = 0,95
AG_GGA_LIT	Intel Core i7-2600	45,1	43,1/45,1 = 0,95
ILS_CMS	Intel Core i7-2600	45,1	43,1/45,1 = 0,95
$\text{MILP}^{+}_{2,1}$	2.67GHz Intel Xeon	36,8	43,1/36,8 = 1,17

Quando forem comparados os resultados obtidos pelo BRKGA_PCMS_BL com os resultados obtidos pelos outros métodos da literatura será calculado o percentual de melhoria da qualidade de solução *gap*(%) dado por:

$$\frac{(\text{MQ literatura} \; - \; \text{MQ BRKGA\_PCMS\_BL})}{\text{MQ literatura}} * 100$$

Note que valores de *gap* negativos indicam que o algoritmo BRKGA_PCMS_BL obteve melhor qualidade de solução quando comparado com a heurística da literatura.

Em todos os experimentos, as médias dos valores de MQ do algoritmo BRKGA_PCMS_BL exibidas nas tabelas foram calculadas como a média de 20 execuções independentes, cada uma com uma diferente semente para o gerador de números pseudo-aleatórios.

4.5.2 Estudo comparativo com heurísticas genéticas da literatura

Selecionou-se quatro abordagens da literatura que procuram resolver o problema de clusterização de módulos de software utilizando algoritmos genéticos (AG) para comparar com o BRKGA_PCMS_BL, a saber: os algoritmos AG_GNE_LIT e AG_GGA_LIT propostos por Pinto (PINTO, 2014), e os algoritmos GNE_LIT e GGA_LIT propostos por Praditwong (PRADITWONG, 2011). A motivação para a escolha de tais heurísticas é que todas utilizam uma abordagem baseada em AG, e os trabalhos representam contribuições importantes na área e realizam experimentos computacionais em subconjuntos de instâncias também utilizadas no presente trabalho.

A Tabela 21 apresenta um resumo das principais características das quatro heurísticas juntamente com a heurística proposta BRKGA_PCMS_BL. A primeira coluna apresenta a identificação da heurística, a segunda coluna apresenta as seguintes informações sobre cada heurística: indicação se a função objetivo é mono-objetivo ou multiobjteivo; o tipo de operador

de recombinação; o método de seleção da população; o tipo de mutação utilizada; se o algoritmo executa pós-processamento ou não; e o número de indivíduos da população.

Tabela 21: Resumo das principais características de algumas heurísticas da literatura baseadas em AG e do BRKGA_PCMS_BL.

Heurística	Característica
AG_GNE_LIT	Mono-objetivo/ Recombinação: *two point crossover* / Seleção: torneio/ Mutação: uniforme/ Pós-processamento: sim/ População: $10 * n$
AG_GGA_LIT	Mono-objetivo/ Recombinação: agrupamento/ Seleção: torneio / Mutação: uniforme / Pós-processamento: sim/ População: $10 * n$
BRKGA_PCMS_BL	Mono-objetivo/ Recombinação: *parametrized uniform crossover* / Seleção: elitista/ Mutação: imigração / Pós-processamento: sim/ População: 100
GNE_LIT	Mono-objetivo/ Recombinação: *one point crossover* / Seleção: *binary tournament*/ Mutação: *one point mutation*/ Pós-processamento: não/ População: $10 * n$
GGA_LIT	Mono-objetivo/ Recombinação: GGA / Seleção: *binary tournament*/ Mutação: *one point mutation* / Pós-processamento: não/ População: $10 * n$

As Tabelas 22, 23, 24 e 25 apresentam a comparação da heurística BRKGA_PCMS_BL com as quatro heurísticas genéticas definidas anteriormente para 42 e 16 instâncias reais da literatura, em relação à qualidade da solução que elas produzem (médias dos MQs) e em relação à eficiência (tempo de execução). A primeira coluna destas tabelas apresenta a identificação das instâncias, a segunda e a terceira colunas reportam as médias de MQ e de tempo da heurística BRKGA_PCMS_BL. A quarta e a quinta coluna apresentam as médias de MQ e de tempo das heurísticas da literatura. Por fim, a última coluna *Gap*(%) apresenta o percentual de melhoria da qualidade de solução (MQ) obtido pela heurística genética da literatura em relação ao BRKGA_PCMS_BL. Note que valores de *gap* negativos indicam que o Algoritmo BRKGA_PCMS_BL obteve melhor qualidade de solução quando comparado com a outra heurística da literatura. Em todas as tabelas, a penúltima linha mostra a média do *gap* e a última linha mostra o tempo computacional total. Os valores em negrito representam as melhores médias da instância.

A Tabela 22 apresenta a comparação entre a heurística BRKGA_PCMS_BL e a heurística AG_GNE_LIT. É possível notar que o BRKGA_PCMS_BL obteve o mesmo resultado de média de MQ para quatro instâncias (“ispell”, “jodamoney”, “apache” e “jmetal”). Em relação ao resto das instâncias, o BRKGA_PCMS_BL obteve a melhor média de MQ em 22 instâncias (todas as instâncias grandes, cinco instâncias médias e cinco instâncias pequenas), equivalente a 52,3% dos casos, enquanto que o AG_GNE_LIT obteve a melhor média em 16 instâncias (nenhuma instância grande, seis instâncias médias e dez instâncias pequenas), equivalente a 38,09% dos casos.

Considerando os resultados observados na Tabela 22 é possível concluir que o BRKGA_PCMS_BL pôde encontrar melhores soluções para instâncias consideradas grandes (mais de 200 módulos) para a formulação mono-objetiva do problema PCMS quando comparado com o genético AG_GNE_LIT. Além disso, a média dos *gaps* ($-3{,}5\%$) também é favorável ao método BRKGA_PCMS_BL. Entretanto, esta melhoria é obtida com um custo computacional

Tabela 22: Comparação do BRKGA_PCMS_BL com o AG_GNE_LIT para 42 instâncias reais.

	BRKGA_PCMS_BL		AG_GNE_LIT[a]		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)[b]	
mtunis	**2,31306**	3,45745	2,29	**0,11**	-1,01
ispell	**2,34000**	5,7876	**2,34**	**0,18**	0,00
jodamoney	**2,73000**	6,0657	**2,73**	**0,16**	0,00
rcs	2,20989	8,74985	**2,24**	**0,24**	1,35
seemp	**4,65358**	6,42585	4,64	**0,23**	-0,29
apache	**5,74000**	9,4079	**5,74**	**0,36**	0,00
bison	2,65305	12,22625	**2,67**	**0,46**	0,63
udtjava	5,1559	32,4328	**5,24**	**1,57**	1,60
javaocr	8,93922	32,49625	**8,95**	**1,66**	0,13
servlet	**9,79147**	34,3881	9,45	**1,80**	-3,61
pfcda_base	7,23595	38,1665	**7,30**	**2,48**	0,87
forms	8,26526	48,8454	**8,30**	**2,68**	0,41
jscatterplot	10,65829	54,94035	**10,68**	**3,32**	0,20
jfluid	6,46824	69,2245	**6,51**	**4,67**	0,64
jxlsscore	**9,57664**	68,5465	9,30	**4,53**	-2,97
grappa	12,52739	58,21745	**12,64**	**4,72**	0,89
jpassword	**10,46543**	99,2996	10,28	**6,93**	-1,81
junit	10,88078	74,1388	**11,06**	**7,31**	1,63
xmldom	**10,86435**	101,69005	10,84	**11,76**	-0,23
tinitym	12,12799	136,6044	**12,31**	**21,68**	1,48
jkarioscope	18,77304	143,08285	**18,90**	**22,04**	0,68
gae_plugin	17,25838	146,5015	**17,29**	**23,12**	0,18
javacc	10,33926	246,19405	**10,43**	**35,49**	0,86
javageom	13,49714	488,73615	**13,67**	**61,65**	1,27
incl	13,50726	276,1709	**13,53**	**41,37**	0,17
jdendogram	**25,66695**	309,85555	25,58	**47,58**	-0,34
xmlapi	**18,67925**	296,59285	18,49	**49,63**	-1,03
jmetal	**11,87**	469,80605	11,87	**69,54**	0,00
dom4j	**18,5571**	538,89135	17,97	**71,79**	-3,27
pdf_renderer	**22,18628**	437,9442	21,19	**69,67**	-4,71
jung_graph	**31,50621**	413,82185	30,16	**78,51**	-4,46
jconsole	**25,97398**	528,4879	24,99	**102,32**	-3,93
jung_visualization	**21,33284**	651,7443	20,16	**102,40**	-5,81
pfcda_swing	**28,0218**	734,0917	26,25	**155,77**	-6,74
jpassword_full	**27,54134**	958,92285	24,74	**203,67**	-11,32
jml	**16,56364**	1235,7595	15,28	**220,76**	-8,40
notepad_full	**28,40743**	1129,18265	25,47	**282,24**	-11,53
porrmans	**33,02675**	1301,9806	28,79	**281,53**	-14,71
log4j	**31,19113**	1313,4396	26,71	**294,78**	-16,77
jtreeview	**47,23018**	1386,92315	39,28	**354,72**	-20,23
jace	**26,17247**	1798,54725	23,16	**422,43**	-13,00
javaws	**36,68585**	2434,17695	29,73	**576,15**	-23,29
Média *gap*					-3,5
Tempo total	-	18144,14225	-	3625,83	-

[a]Resultados extraídos da coluna AG_GNE das Tabelas 18, 20, 22, 24, 25 e 26 (PINTO, 2014).

[b]Tempos computacionais multiplicados pelo fator de escala de 0,95.

alto, aproximadamente cinco vezes mais lento que o AG_GNE_LIT.

A Tabela 23 apresenta a comparação entre a heurística BRKGA_PCMS_BL e a heurística AG_GGA_LIT. É possível notar que o BRKGA_PCMS_BL obteve o mesmo resultado de média de MQ em duas instâncias pequenas ("mtunis" e "seemp"). Em relação às outras instâncias, o BRKGA_PCMS_BL superou a média de MQ em cinco casos (duas instâncias pequenas, uma média e duas grandes). Para as outras 35 instâncias, o AG_GGA_LIT obteve melhores resultados de média de MQ.

Considerando os resultados observados na Tabela 23 e a quantidade de vezes que um método superou o outro, é possível verificar que o AG_GGA_LIT pôde encontrar melhores soluções para instâncias pequenas, médias e grandes para a formulação mono-objetiva do problema PCMS quando comparado com o BRKGA_PCMS_BL.

Em relação ao tempo computacional gasto por ambas heurísticas, o BRKGA_PCMS_BL possui uma média de tempo aproximadamente 15,07% superior, sendo que nas quatro instâncias maiores seu tempo computacional foi inferior ao tempo computacional do AG_GGE_LIT.

A Tabela 24 apresenta a comparação entre a heurística BRKGA_PCMS_BL e a heurística GGA_LIT. Nota-se que o BRKGA_PCMS_BL obteve maiores médias de MQ em 12 das 16 instâncias (todas as sete médias e em cinco das nove instâncias pequenas), equivalente a 75% dos casos. Em relação às outras instâncias, o GGA_LIT superou a média de MQ em três casos (três instâncias pequenas), sendo que em apenas uma instância, "ispell", os resultados foram iguais.

Considerando os resultados observados na Tabela 24 e a quantidade de vezes que um método superou o outro, é possível concluir que o BRKGA_PCMS_BL pôde encontrar melhores soluções para instâncias pequenas, médias e grandes para a formulação mono-objetiva do problema PCMS do que o genético GGA_LIT. A média dos *gaps* ($-1{,}98\%$) também é favorável ao método BRKGA_PCMS_BL.

Não foi possível realizar comparações de eficiência pois os tempos de execução dos dois métodos não são comparáveis, visto que o processador utilizado nos experimentos com GGA_LIT não estava disponível na referência (PRADITWONG, 2011).

A Tabela 25 apresenta a comparação entre a heurística BRKGA_PCMS_BL e a heurística GNE_LIT. Nota-se que o BRKGA_PCMS_BL obteve maiores médias de MQ em 14 das 16 instâncias (87,5% dos casos). Em relação às outras instâncias o GNE_LIT superou a média de MQ em dois casos (duas instâncias pequenas).

Considerando os resultados observados na Tabela 25 e a quantidade de vezes que um método superou o outro, é possível concluir que o BRKGA_PCMS_BL pôde encontrar melhores soluções para instâncias pequenas, médias e grandes para a formulação mono-objetiva do problema PCMS do que o genético GNE_LIT. Além disso, a média dos *gaps* ($-29{,}01\%$) também é favorável ao método BRKGA_PCMS_BL.

Tabela 23: Comparação do BRKGA_PCMS_BL com a heurística AG_GGA_LIT para 42 instâncias reais.

	BRKGA_PCMS_BL		AG_GGA_LIT[a]		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)[b]	
mtunis	**2,31000**	3,45745	**2,31**	**0,10**	0,00
ispell	2,34163	5,7876	**2,36**	**0,18**	0,78
jpdamoney	2,73802	6,0657	**2,75**	**0,27**	0,44
rcs	2,20989	8,74985	**2,25**	**0,33**	1,79
seemp	**4,65000**	6,42585	**4,65**	**0,45**	0,00
apache	5,74344	9,4079	**5,77**	**0,71**	0,47
bison	2,65305	12,22625	**2,69**	**0,67**	1,38
udtjava	5,1559	32,4328	**5,28**	**3,23**	2,36
javaocr	8,93922	32,49625	**9,02**	**3,75**	0,90
servlet	**9,79147**	34,3881	9,50	**4,63**	-3,07
pfcda_base	7,23595	38,1665	**7,32**	**5,85**	1,15
forms	8,26526	48,8454	**8,32**	**6,28**	0,66
jscatterplot	10,65829	54,94035	**10,74**	**8,81**	0,77
jfluid	6,46824	69,2245	**6,54**	**12,96**	1,10
jxlscore	**9,57664**	68,5465	9,29	**13,51**	-3,09
grappa	12,52739	58,21745	**12,69**	**12,46**	1,29
jpassword	**10,46543**	99,2996	10,34	**22,19**	-1,22
junit	10,88078	74,1388	**11,08**	**19,95**	1,8
xmldom	10,86435	101,69005	**10,88**	**37,79**	0,15
tinytim	12,12799	136,6044	**12,45**	**59,57**	2,59
jkaryoscope	18,77304	143,08285	**18,96**	**61,11**	0,99
gae_plugin	17,25838	146,5015	**17,29**	**69,88**	0,18
javacc	10,33926	246,19405	**10,62**	**102,09**	0,29
javageom	13,49714	488,73615	**14,01**	**160,88**	3,67
incl	13,50726	276,1709	**13,58**	**167,64**	0,54
jdendogram	25,66695	309,85555	**26,03**	**163,74**	1,40
xmlapi	18,67925	296,59285	**18,97**	**191,71**	1,54
jmetal	11,87508	469,80605	**12,39**	**223,95**	4,16
dom4j	18,55710	538,89135	**18,77**	**223,74**	1,14
pdf_renderer	**22,18628**	437,9442	21,82	**263,45**	-1,68
jung_graph	31,50621	413,82185	**31,55**	**287,18**	0,14
jconsole	25,97398	528,4879	**26,43**	**293,59**	1,73
jung_visualization	**21,33284**	651,7443	21,05	**385,89**	-1,35
pfcda_swing	28,0218	734,0917	**28,85**	**655,02**	2,88
jpassword_full	27,54134	958,92285	**27,87**	**811,29**	1,18
jml	16,56364	1235,7595	**17,35**	**857,23**	4,54
notepad_full	28,40743	**1129,18265**	**29,49**	1205,13	3,68
porrmans	33,02675	1301,9806	**34,06**	**1275,42**	3,04
log4j	31,19113	**1313,4396**	**31,49**	1375,03	0,95
jtreeview	47,23018	**1386,92315**	**47,55**	1730,28	0,68
jace	26,17247	**1798,54725**	**26,59**	2076,23	1,58
javaws	36,68585	**2434,17695**	**38,19**	3005,89	3,94
Média *gap*					1,13
Tempo total		18144,14225		15767,30	

[a]Resultados extraídos da coluna AG_GGA das Tabelas 18, 20, 22, 24, 25 e 26 (PINTO, 2014).

[b]Tempos computacionais multiplicados pelo fator de escala de 0,95.

Tabela 24: Comparação do BRKGA_PCMS_BL com a heurística GGA_LIT para 16 instâncias da literatura.

	BRKGA_PCMS_BL		GGA_LIT[a]		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)[b]	
mtunis	**2,31306**	3,45745	2,230	-	-3,73
ispell	**2,34000**	5,78760	2,340	-	0,00
rcs	2,20989	8,74985	**2,232**	-	0,99
bison	**2,65305**	12,22625	2,231	-	-18,98
grappa	**12,52739**	58,21745	12,450	-	-0,63
incl	**13,50726**	276,16775	13,139	-	-2,80
icecast	**2,68316**	70,7605	2,665	-	-0,69
gnupg	**7,01598**	115,8809	6,874	-	-2,07
inn	7,73672	131,48605	**7,911**	-	2,21
bitchx	4,19485	222,6751	**4,252**	-	1,35
xntp	**8,16141**	128,4259	8,111	-	-0,63
exim	**6,3808**	243,948	6,251	-	-2,08
mod_ssl	**9,91303**	278,0382	9,831	-	-0,84
ncurses	**11,45333**	246,24475	11,452	-	-0,02
lynx	**4,66851**	400,2485	4,521	-	-3,27
nmh	**8,80795**	1257,4422	8,770	-	-0,44
Média *gap*					-1,98
Tempo total		3459,75645			

[a]Resultados extraídos da coluna GGA da Tabela 3 (PRADITWONG, 2011).
[b]Os tempos computacionais não foram reportados pois eles não são comparáveis.

Mais uma vez não foi possível realizar comparações de eficiência pois os tempos de execução dos dois métodos não são comparáveis, visto que o processador utilizado nos experimentos com GNE_LIT não estava disponível na referência (PRADITWONG, 2011).

Por fim, conclui-se que a heurística proposta BRKGA_PCMS_BL quando comparada com abordagens de AG da literatura possui, em todos os experimentos realizados, uma eficiência inferior, sendo na média (apenas dois dos experimentos possuíam o tempo computacional nas referências) 283,5% mais lenta. Em relação à qualidade de suas soluções (eficácia), considerando os quatro experimentos genéticos realizados, a média dos *gaps* foi favoravel à heurística proposta em três dos quatro experimentos realizados (na comparação com a heurística AG_GGA_LIT a média dos *gaps* foi de 1,13%). Entretanto, tal resultado não é suficiente para afirmar que um método é superior em relação ao outro. Isto pois, caso os demais métodos da literatura tivessem disponível o mesmo tempo de processamento que a heurística BRKGA_PCMS_BL usou, eventualmente a qualidade das soluções encontradas por estes métodos poderia ser tão boa quanto, ou melhor que as encontradas pelo BRKGA_PCMS_BL.

Tabela 25: Comparação do BRKGA_PCMS_BL com a heurística GNE_LIT.

	BRKGA_PCMS_BL		GNE_LIT[a]		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)[b]	
mtunis	**2,31306**	3,45745	2,290	-	-1,01
ispell	2,34163	5,78760	**2,350**	-	0,36
rcs	2,20989	8,74985	**2,260**	-	2,22
bison	**2,65305**	12,22625	2,290	-	-15,86
grappa	**12,52739**	58,21745	10,830	-	-15,67
incl	**13,50726**	276,1709	8,140	-	-65,93
icecast	**2,68316**	70,7605	2,668	-	-0,57
gnupg	**7,01598**	115,8809	6,072	-	-15,55
inn	**7,73672**	131,48605	6,296	-	-22,89
bitchx	**4,19485**	222,6751	3,565	-	-17,67
xntp	**8,16141**	128,4259	6,029	-	-35,37
exim	**6,3808**	243,948	4,848	-	-31,62
mod_ssl	**9,91303**	278,0382	6,860	-	-44,51
ncurses	**11,45333**	246,24475	7,439	-	-53,97
lynx	**4,66851**	400,2485	3,037	-	-53,73
nmh	**8,80795**	1257,4422	4,576	-	-92,49
Média *gap*					-29,01
Tempo total		3459,7596			

[a]Resultados extraídos da coluna GNE da Tabela 3 (PRADITWONG, 2011).
[b]Os tempos computacionais não foram reportados pois eles não são comparáveis.

4.5.3 Estudo comparativo com outras heurísticas da literatura

A próxima comparação com heurísticas da literatura aborda duas heurísticas, a saber: a heurística ILS_CMS (PINTO, 2014), baseada na metaheurística busca local iterada, em inglês, *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003), e o algoritmo exato $\text{MILP}^{+}_{2,1}$ (KÖHLER; FAMPA; ARAUJO, 2013), que formula o PCMS como um problema de soma de funções lineares fracionais (SOLF) (CHEN et al., 2005) no qual é aplicado um procedimento de linearização para reformular o SOLF como um problema de programação mista inteira linear, em inglês *Mixed-Integer Linear Programming* (MILP).

As Tabelas 26 e 27 apresentam a comparação da heurística BRKGA_PCMS_BL com estes dois métodos. A primeira coluna informa a identificação da instância, a segunda e a terceira colunas reportam as médias de MQ e de tempo da heurística BRKGA_PCMS_BL. A quarta e a quinta colunas apresentam as médias de MQ e de tempo das heurísticas da literatura. A última coluna *Gap*(%) apresenta o percentual de melhoria da qualidade de solução (MQ) obtido pela heurística da literatura em relação ao BRKGA_PCMS_BL. Note que valores de *gap* negativos indicam que o algoritmo BRKGA_PCMS_BL obteve melhor qualidade de solução quando comparado com a heurística da literatura. A penúltima linha mostra a média do *gap* e a última linha mostra o tempo computacional total.

Tabela 26: Comparação do BRKGA_PCMS_BL com o algoritmo exato MILP$^{+}_{2,1}$.

	BRKGA_PCMS_BL		MILP_2,1^{+}		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ^{a}	Tempo (s.)^{a}	
small	**1,83330**	0,46905	**1,8333**	**0,02**	0,00
compiler	1,50505	1,58460	**1,5065**	**1,07**	0,10
lab4	**3,40000**	1,89740	**3,4000**	**0,05**	0,00
nos	1,67705	**2,60370**	**1,6775**	4,76	0,03
lslayout	1,84008	2,61875	**1,8613**	**1,30**	1,15
boxer	3,05290	2,74835	**3,1011**	**0,27**	1,56
mtunis	2,31306	**3,45745**	**2,3145**	39,28	0,07
spdb	**5,58970**	3,70450	**5,5897**	**0,03**	0,00
bunch	**2,40600**	4,94270	**2,4060**	**1,44**	0,00
ispell	2,33603	**5,78760**	**2,3639**	337,47	1,18
ciad	2,85125	**5,54875**	**2,8513**	14,43	0,01
modulizer	**2,75790**	**5,86480**	**2,7579**	10,07	0,00
rcs	2,20989	**8,74960**	**2,2775**	12870,00	2,97
star	3,80452	**9,3913**	**3,8321**	12870,00	0,72
bison	2,65305	**2,65305**	**2,6999**	12870,00	1,74
Média *gap*					0,63
Tempo total		62,0216		39020,19	

[a] Resultados extraídos das colunas z e "Time(Sec)" da Tabela 5 (KÖHLER; FAMPA; ARAUJO, 2013). Tempos computacionais de MILP_2,1^{+} multiplicados pelo fator de escala de 1,17. As instâncias "rcs", "star" e "bison" não tiveram o tempo de execução exato reportado em (KÖHLER; FAMPA; ARAUJO, 2013), foi apenas informado que executaram em mais de 11.000 segundos. Para efeito de comparação, o tempo de 11.000 segundos foi usado nesta dissertação.

A Tabela 26 apresenta a comparação entre a heurística BRKGA_PCMS_BL e o algoritmo exato MILP$^{+}_{2,1}$. É possível notar que, para cinco instâncias ("small", "lab4", "spdb", "bunch" e "modulizer"), a média de MQ obtida pelo BRKGA_PCMS_BL é igual ao resultado encontrado pelo MILP$^{+}_{2,1}$, que, nestes cinco casos, é comprovadamente o valor da solução ótima. Em relação às 11 instâncias restantes, o BRKGA_PCMS_BL obteve uma média de MQ inferior ao MILP$^{+}_{2,1}$. A média dos *gaps* é favorável ao método MILP$^{+}_{2,1}$ em $0{,}63\%$.

Considerando os resultados observados na Tabela 26, é possível concluir que o MILP$^{+}_{2,1}$ pôde encontrar melhores soluções para instâncias pequenas (até 100 módulos) para a formulação mono-objetiva do problema PCMS quando comparado com o BRKGA_PCMS_BL. De todas as 15 instâncias, o BRKGA_PCMS_BL obteve o mesmo resultado de MQ em cinco instâncias sendo superado pelo MILP$^{+}_{2,1}$ nas outras dez instâncias. Entretanto, essa melhoria é obtida com um custo computacional alto: o método MILP$^{+}_{2,1}$ é aproximadamente 629 vezes mais lento do que o BRKGA_PCMS_BL.

A Tabela 27 apresenta a comparação entre a heurística BRKGA_PCMS_BL e a heurística ILS_CMS. É possível notar que o BRKGA_PCMS_BL obteve o mesmo resultado de média

Tabela 27: Comparação do BRKGA_PCMS_BL com a heurística ILS_CMS.

	BRKGA_PCMS_BL		ILS_CMS[a]		*Gap*(%)
Instância	MQ	Tempo (s.)	MQ	Tempo (s.)	
mtunis	**2,31000**	3,45745	**2,31**	**0,01**	0,00
ispell	2,34163	5,7876	**2,35**	**0,01**	0,36
jodamoney	2,73802	6,0657	**2,75**	**0,01**	0,44
rcs	2,20989	8,74985	**2,24**	**0,02**	1,35
seemp	**4,65000**	6,42585	**4,65**	**0,01**	0,00
apache	5,74344	9,4079	**5,77**	**0,02**	0,47
bison	2,65305	12,22625	**2,69**	**0,03**	1,38
udtjava	5,1559	32,4328	**5,28**	**0,06**	2,36
javaocr	8,93922	32,49625	**9,00**	**0,05**	0,68
servlet	**9,79147**	34,3881	9,47	**0,05**	-3,40
pfcda_base	7,23595	38,1665	**7,33**	**0,08**	1,29
forms	8,26526	48,8454	**8,33**	**0,11**	0,78
jscatterplot	10,65829	54,94035	**10,74**	**0,10**	0,77
jfluid	6,46824	69,2245	**6,58**	**0,15**	1,70
jxlsscore	**9,57664**	68,5465	9,30	**0,14**	-2,98
grappa	12,52739	58,21745	**12,70**	**0,13**	1,36
jpassword	**10,46543**	99,2996	10,29	**0,17**	-1,71
junit	10,88078	74,1388	**11,09**	**0,17**	1,89
xmldom	**10,86000**	101,69005	**10,86**	**0,21**	0,00
tinitym	12,12799	136,6044	**12,51**	**0,37**	3,06
jkarioscope	18,77304	143,08285	**18,98**	**0,34**	1,10
gae_plugin	17,25838	146,5015	**17,28**	**0,34**	0,13
javacc	10,33926	246,19405	**10,60**	**0,53**	2,46
javageom	13,49714	488,73615	**14,07**	**0,92**	4,08
incl	13,50726	276,1709	**13,61**	**0,52**	0,76
jdendogram	25,66695	309,85555	**26,07**	**0,55**	1,55
xmlapi	18,67925	296,59285	**18,99**	**0,53**	1,64
jmetal	11,87508	469,80605	**12,41**	**0,86**	4,32
dom4j	18,5571	538,89135	**18,83**	**0,79**	1,45
pdf_renderer	**22,18628**	437,9442	21,85	**0,73**	-1,54
jung_graph	31,50621	413,82185	**31,52**	**0,67**	0,05
jconsole	25,97398	528,4879	**26,51**	**0,91**	2,03
jung_visualization	**21,33284**	651,7443	20,88	**0,91**	-2,17
pfcda_swing	28,0218	734,0917	**28,99**	**1,10**	3,34
jpassword_full	27,54134	958,92285	**27,89**	**1,54**	1,26
jml	16,56364	1235,7595	**17,40**	**1,82**	4,81
notepad_full	28,40743	1129,18265	**29,48**	**1,79**	3,64
porrmans	33,02675	1301,9806	**34,13**	**1,67**	3,24
log4j	31,19113	1313,4396	**31,43**	**1,75**	0,77
jtreeview	47,23018	1386,92315	**47,59**	**1,87**	0,76
jace	26,17247	1798,54725	**26,63**	**2,74**	1,72
javaws	36,68585	2434,17695	**38,27**	**2,82**	4,14
Média *gap*					1,16
Tempo total		18144,14225	27,61		

[a]Resultados extraídos da coluna ILS_CMS das Tabelas 18, 20, 22, 24, 25 e 26 (PINTO, 2014). Tempos computacionais do ILS_CMS[a] multiplicados pelo fator de escala de 0,95.

de MQ para três instâncias pequenas (“seemp”,“mtunis” e ”xml dom”). Em relação ao resto das instâncias, o BRKGA_PCMS_BL obteve a melhor média de MQ em cinco instâncias (uma instância grande, duas instâncias médias e duas instâncias pequenas) enquanto que o ILS_CMS obteve a melhor média em 34 (80% dos casos).

Considerando os resultados observados na Tabela 27, é possível concluir que o ILS_CMS

pôde encontrar melhores soluções para instâncias consideradas pequenas, médias e grandes para a formulação mono-objetiva do problema PCMS quando comparado com o BRKGA_PCMS_BL, inclusive com um custo computacional aproximadamente 657 vezes inferior.

4.5.4 Estudo comparativo com os melhores resultados da literatura

A próxima comparação visa analisar eficiência e eficácia do BRKGA_PCMS_BL quando comparando seus resultados com os melhores resultados documentados do problema de clusterização de módulos de software para 113 instâncias reais da literatura. A primeira coluna da Tabela 28 apresenta a identificação da instância, a segunda coluna reporta a quantidade de módulos da instância, a terceira e quarta colunas reportam as médias de MQ e de tempo da heurística BRKGA_PCMS_BL. A quinta e sexta colunas apresentam as médias de MQ e de tempo das heurísticas que obtiveram os melhores resultados de MQ da literatura. A próxima coluna *Gap*(%) apresenta o percentual de melhoria da qualidade de solução (MQ) obtido pelo método da literatura em relação ao BRKGA_PCMS_BL. Note que valores de *gap* negativos indicam que o algoritmo BRKGA_PCMS_BL obteve melhor qualidade de solução quando comparado com outro método da literatura. Por fim, a última coluna exibe a referência da literatura que encontrou o melhor valor de MQ conhecido. A penúltima linha mostra a média do *gap* e a última linha mostra o tempo computacional total. Um "*" é colocado quando, para determinada instância, não tiver sido encontrado nem o MQ nem o tempo de processamento do mesmo.

É possível notar que o BRKGA_PCMS_BL obteve o mesmo resultado de média de MQ para dez instâncias pequenas (8,84%). Em relação ao resto das instâncias, o BRKGA_PCMS_BL obteve a melhor média de MQ em dez instâncias (8,84% dos casos), enquanto que os melhores métodos da literatura obtiveram a melhor média nas 93 outras instâncias (82,30% dos casos).

Tabela 28: Comparação do BRKGA_PCMS_BL com os melhores resultados da literatura para 113 instâncias.

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
squid	2	1	0,08175	*	*	*	-
small	6	**1,83330**	0,46905	**1,8333**	**0,0100**	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
compiler	13	1,50505	1,5846	**1,5065**	**0,9100**	0,1	(KÖHLER; FAMPA; ARAUJO, 2013)
jstl	15	5,00000	2,1772	*	*	*	-
lab4	15	**3,40000**	1,8975	**3,4000**	**0,0400**	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-ping	15	**1,00000**	1,6995	**1,0000**	**0,0000**	0	(KÖHLER; FAMPA; ARAUJO, 2013)
nss_ldap	15	**1,0000**	2,0518	**1,0000**	**0,0100**	0	(KÖHLER; FAMPA; ARAUJO, 2013)
nos	16	1,67717	**2,6037**	**1,6775**	4,06	0,02	(KÖHLER; FAMPA; ARAUJO, 2013)
lslayout	17	1,84008	2,61875	**1,8613**	**1,1100**	1,15	(KÖHLER; FAMPA; ARAUJO, 2013)
boxer	18	3,05896	2,74835	**3,1011**	**0,2300**	1,36	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-tftpd	18	**1,14420**	2,355	**1,14420**	**0,01000**	0	(KÖHLER; FAMPA; ARAUJO, 2013)
sharutils	19	2,54766	2,82195	**2,5492**	**0,4300**	0,07	(KÖHLER; FAMPA; ARAUJO, 2013)
mtunis	20	2,31306	**3,45745**	**2,3145**	33,57	0,07	(KÖHLER; FAMPA; ARAUJO, 2013)
spdb	21	**5,58970**	3,7045	**5,5897**	**0,0200**	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
xtell	22	**2,00520**	3,824	**2,0052**	0,36	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
bunch	24	**2,40600**	4,9427	**2,4060**	**1,2300**	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
ispell	24	2,34163	**5,7876**	**2,3639**	288,43	0,94	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-inetd	24	1,31205	3,8698	**1,3121**	**0,0100**	0,01	(KÖHLER; FAMPA; ARAUJO, 2013)
nanoxml	25	3,78453	5,372	3,82	**0,0100**	0,93	(PINTO, 2014)
ciald	26	2,85125	**5,54875**	**2,8513**	12,33	0,06	(KÖHLER; FAMPA; ARAUJO, 2013)
jodamoney	26	2,73802	6,0657	**2,7500**	**0,0100**	0,44	(PINTO, 2014)
bootp	27	2,19502	5,74715	**2,1985**	**1,9800**	0,16	(KÖHLER; FAMPA; ARAUJO, 2013)

Tabela 28: (continuação)

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
jxlsreader	27	3,58602	6,1215	**3,6000**	**0,0100**	0,39	(PINTO, 2014)
modulizer	27	**2,7579**	**5,8649**	**2,7579**	8,6	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
sysklogd-1	28	1,69171	6,2146	**1,7105**	**1,4700**	8,44	(KÖHLER; FAMPA; ARAUJO, 2013)
telnetd	28	**1,84750**	5,7355	**1,8475**	**0,5900**	0,00	(KÖHLER; FAMPA; ARAUJO, 2013)
crond	29	2,29658	6,99485	**2,3030**	**3,3000**	0,28	(KÖHLER; FAMPA; ARAUJO, 2013)
netkit-ftp	29	1,76443	6,48685	**1,7680**	**0,5600**	0,21	(KÖHLER; FAMPA; ARAUJO, 2013)
rcs	29	2,20989	8,74985	**2,2775**	11000,00	2,98	(KÖHLER; FAMPA; ARAUJO, 2013)
seemp	30	4,65358	6,42585	**4,7000**	**0,2000**	0,99	(BARROS, 2014)
dhcpd-2	31	3,48408	**8,1431**	**3,4944**	16,25	0,3	(KÖHLER; FAMPA; ARAUJO, 2013)
cyrus-sasl	32	3,25151	7,7943	**3,2518**	**0,8200**	0,01	(KÖHLER; FAMPA; ARAUJO, 2013)
tcsh	32	1,20233	7,7892	**1,2141**	0,4800	**0,97**	(KÖHLER; FAMPA; ARAUJO, 2013)
micq	33	2,1338	11,0071	**2,1680**	**4,4500**	1,58	(KÖHLER; FAMPA; ARAUJO, 2013)
apache	36	5,74344	9,4079	**5,80000**	**0,3000**	0,98	(BARROS, 2014)
star	36	3,80452	**9,3915**	**3,8321**	11000,00	0,96	(KÖHLER; FAMPA; ARAUJO, 2013)
bison	37	**2,65305**	12,22625	**2,6999**	11000,00	1,88	(KÖHLER; FAMPA; ARAUJO, 2013)
cia	38	3,72153	26,9848	*	*	*	-
stunnel	38	2,49062	9,95355	**2,5261**	**0,5700**	1,41	(KÖHLER; FAMPA; ARAUJO, 2013)
minicom	40	2,53551	**15,85105**	**2,5759**	776,53	1,57	(KÖHLER; FAMPA; ARAUJO, 2013)
mailx	41	3,15947	**19,18465**	**3,2353**	11000,0000	2,5	(KÖHLER; FAMPA; ARAUJO, 2013)
screen	41	2,20475	17,75295	*	*	*	-
dot	43	2,78125	19,32215	*	*	*	-
slang	45	4,65865	18,8854	**4,6794**	11000,00	0,45	(KÖHLER; FAMPA; ARAUJO, 2013)
slrn	45	2,34138	22,5148	**2,3928**	11000,00	2,15	(KÖHLER; FAMPA; ARAUJO, 2013)

Tabela 28: (continuação)

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
net-tools	48	4,29533	18,86255	**4,3159**	11000,00	0,48	(KÖHLER; FAMPA; ARAUJO, 2013)
wu-ftpd-1	50	2,40828	**23,051**	**2,4437**	312,71	1,46	(KÖHLER; FAMPA; ARAUJO, 2013)
joe	51	3,25582	34,10205	**3,2926**	11000,00	2,56	(KÖHLER; FAMPA; ARAUJO, 2013)
imapd-1	53	3,51997	**24,97875**	**3,6250**	206,8	2,9	(KÖHLER; FAMPA; ARAUJO, 2013)
wu-ftpd-3	54	3,35154	27,737	**3,3953**	11000,00	1,29	(KÖHLER; FAMPA; ARAUJO, 2013)
udt-java	56	5,1559	32,4318	**5,3000**	**1,0000**	2,72	(BARROS, 2014)
javaocr	58	8,93922	32,49625	**9,0200**	**3,9400**	0,9	(PINTO, 2014)
dhcpd-1	59	3,94873	61,00195	**3,9359**	11000,00	0,97	(KÖHLER; FAMPA; ARAUJO, 2013)
icecast	60	2,68316	70,76195	**2,7544**	11000,00	2,59	(KÖHLER; FAMPA; ARAUJO, 2013)
pfcda_base	60	7,23595	38,1665	**7,3300**	**0,0600**	1,29	(PINTO, 2014)
servletapi	61	**9,79147**	34,3881	9,5	**4,8700**	-3,07	(BARROS, 2014)
php	62	5,22628	**29,3542**	**5,3242**	226,11	1,84	(KÖHLER; FAMPA; ARAUJO, 2013)
bunch2	65	7,69024	31,66875	*	*	*	-
forms	68	8,26526	48,8454	**8,3300**	**0,1100**	0,78	(PINTO, 2014)
jscatterplot	74	10,65829	54,94035	**10,7400**	**0,1000**	0,77	(PINTO, 2014)
jxlscore	79	**9,57664**	68,5465	9,3	**0,1400**	-2,98	(PINTO, 2014)
elm-2	81	3,66864	121,2767	*	*	*	-
jfluid	81	6,46824	69,2245	**6,5800**	**0,1500**	1,7	(PINTO, 2014)
grappa	86	12,52739	58,21745	**12,7000**	**0,1300**	1,36	(PINTO, 2014)
elm-1	88	4,15609	166,1613	*	*	*	-
gnupg	88	7,01598	115,88375	*	*	*	-
inn	90	7,73672	131,48605	**7,9110**	*	1,02	(PRADITWONG, 2011)
bash	92	5,7155	129,8472	*	*	*	-

Tabela 28: (continuação)

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
jpassword	96	**10,46543**	99,2996	10,34	**23,3500**	-1,22	(PINTO, 2014)
bitchx	97	4,19485	222,6751	**4,2670**	*	1,7	(PRADITWONG, 2011)
junit	100	10,88078	74,1388	**11,1000**	**5,1000**	1,98	(BARROS, 2014)
xntp	111	8,16141	128,4259	**8,786**	*	7,10	(CAMI; SRINIVASULU, 2012)
acqcigna	114	16,46214	90,85955	*	*	*	-
bunch_2	116	13,39931	112,4164	-	-	-	-
exim	118	**6,3808**	243,948	6,361	*	-0,32	(PRADITWONG; HARMAN; YAO, 2011)
xmldom_n	118	10,86435	101,69005	**10,9000**	**6,6000**	0,33	(BARROS, 2014)
cia++	124	15,40047	126,7316	*	*	*	-
tinytim	129	12,12799	136,6044	**12,5100**	**0,2800**	3,06	(PINTO, 2014)
mod_ssl	135	**9,91303**	278,0382	9,831	*	-0,84	(PRADITWONG, 2011)
jkaryoscope	136	18,77304	143,08285	**18,9800**	**0,3500**	1,1	(PINTO, 2014)
ncurses_n	138	**11,53391**	246,24475	11,452	*	-0,72	(PRADITWONG, 2011)
gae_plugin_core	139	17,25838	152,02095	**17,3200**	**0,2500**	0,42	(PINTO, 2014)
lynx	148	4,66851	400,2485	**4,6940**	*	0,55	(PRADITWONG; HARMAN; YAO, 2011)
javacc	153	10,33926	246,19405	**10,6200**	**107,4600**	2,65	(PINTO, 2014)
lucent	153	59,66874	189,541	*	*	*	*
javageom	171	13,49714	488,73615	**14,0700**	**0,9600**	4,08	(PINTO, 2014)
incl	174	13,50726	276,1709	**13,61**	**0,54**	0,97	(PINTO, 2014)
jdendogram	177	25,66695	309,85555	**26,0700**	**0,5700**	1,55	(PINTO, 2014)
xmlapi	182	18,74182	296,59285	**18,9900**	**0,5500**	1,31	(PINTO, 2014)
jmetal	190	11,87508	469,80605	**12,4100**	**0,9000**	4,32	(PINTO, 2014)
dom4j	195	18,55715	538,89135	**18,8300**	**0,8300**	1,45	(PINTO, 2014)

Tabela 28: (continuação)

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
nmh	198	**8,80795**	1257,4422	8,77	*	-0,44	(PRADITWONG, 2011)
pdf_renderer	199	**22,18628**	437,9442	21,85	**277,3100**	-1,22	(PINTO, 2014)
jung_graph_model	207	31,50621	413,82185	**31,5500**	**302,2900**	0,14	(PINTO, 2014)
jconsole	220	25,97398	528,4879	**26,5100**	**0,9500**	2,03	(PINTO, 2014)
jung_visualization	221	**21,33284**	651,7443	21,05	**406,1900**	-1,35	(PINTO, 2014)
pfcda_swing	252	28,0218	734,0917	**28,9900**	**0,9000**	3,34	(PINTO, 2014)
jpassword2	269	27,54134	1121,8238	**27,8900**	**1,6200**	1,26	(PINTO, 2014)
jml	270	16,56364	1235,7595	**17,4000**	**1,9100**	4,81	(PINTO, 2014)
notelab-full	293	28,40743	**1129,18265**	**29,4900**	1268,55	3,68	(PINTO, 2014)
poormans cms	301	33,02675	1301,9806	**34,1300**	**1,7500**	3,24	(PINTO, 2014)
log4j	305	31,19113	**1313,4396**	**31,4900**	1447,39	0,95	(PINTO, 2014)
jtreeview	320	47,23018	1386,9232	**47,5900**	**1,9600**	0,76	(PINTO, 2014)
bunchall	324	16,14042	1631,9596	*	*	*	*
jace	338	26,17247	1798,5473	**26,6300**	*	1,72	(PINTO, 2014)
javaws	377	36,68585	2434,177	**38,2700**	**2,9600**	4,14	(PINTO, 2014)
swing	413	42,80639	2562,5337	*	*	*	-
lwjgl-2.8.4	453	35,98682	3907,3574	*	*	*	-
ping_libc	481	49,92857	5699,1904	*	*	*	-
y_base	556	55,01983	6073,3116	*	*		-
krb5	558	51,23881	8329,4609	*	*	*	-
apache_ant_taskdef	626	63,93641	8975,1909	*	*	*	-
apache_lucene_core	738	70,76018	13157,2	*	*	*	-
Média *gap*						1,69	

Tabela 28: (continuação)

		BRKGA_PCMS_BL		Melhor Literatura			
Instância	# Módulos	MQ	Tempo (s.)	MQ	Tempo (s.)	*Gap*(%)	Referência
Tempo total			16708,49305		127079,40		

5 Conclusão

O principal objetivo deste trabalho de dissertação de mestrado foi o de apresentar uma heurística, baseada na metaheurística BRKGA, para o problema de clusterização de módulos de software. A heurística proposta BRKGA_PCMS_BL é composta, entre outros, de um codificador, de um decodificador e de uma busca local. A seguir, na Seção 5.1 apresentam-se as principais contribuições do trabalho e, em seguida, na Seção 5.2, as limitações do trabalho e perspectivas de trabalhos futuros.

5.1 Contribuições

No Capítulo 2 apresentou-se um levantamento de 18 referências sobre trabalhos da literatura que abordam o PCMS, sumarizadas na Tabela 1, em que se destaca, para cada referência em ordem cronológica, a abordagem proposta, o tipo de função objetivo utilizada, a quantidade de instâncias utilizadas e o tamanho da maior instância. Para um conjunto de 113 instâncias teste, outro levantamento destaca, na Tabela 28, para cada instância, além das características de quantidade de módulos e dependências, o valor da solução ótima, quando disponível. Acredita-se que tais levantamentos possam ser úteis para futuros trabalhos relacionados ao PCMS.

Os três primeiros estudos experimentais visam caracterizar a heurística proposta e foram realizados em um subconjunto de 12 instâncias. O primeiro estudo, documentado na Seção 4.4.1, considera 13 diferentes configurações do algoritmo proposto, variando-se os valores dos parâmetros do *framework* BRKGA. Seus resultados indicaram que as configurações do BRKGA que utilizam uma estratégia de modificação dinâmica de dois parâmetros do *framework* (o percentual de mutantes, p_m, e o percentual de elites, p_e) foi mais eficaz do que a estratégia que usa valores fixos (a estratégia "Combinado" obteve os melhores resultados em 11 das 12 instâncias). O segundo estudo experimental, apresentado na Seção 4.4.2, considera a melhor configuração do estudo anterior e investiga o comportamento do algoritmo variando-se, além do percentual de vizinhos (p_v), as escolhas para os principais componentes do método BRKGA, que são os decodificadores e os codificadores, por meio de 24 configurações distintas. A principal contribuição deste experimento é a conclusão da superioridade da utilização de decodificadores compostos, ou seja, a utilização do método LBF+BF, que escolhe a partir de uma probabilidade de 50% a estratégia LBF ou a estratégia BF, em detrimento da utilização de um codificador simples. Percebe-se que, em oito instâncias, este método conseguiu obter as melhores médias de MQ apenas com cinco configurações de decodificadores compostos (E2C4, E2C9, E2C14, E2C19, E2C24) enquanto que as outras 19 configurações, que utilizam os decodificadores simples (BF, FF, LBF e WF) conseguiram obter as melhores médias apenas em seis instâncias. Em seguida, na Seção 4.4.3, aplicou-se uma busca local à melhor configuração do segundo

estudo, resultando na versão final do algoritmo proposto BRKGA_PCMS_BL.

Para avaliar eficiência e eficácia do algoritmo proposto, o BRKGA_PCMS_BL foi comparado, na Seção 4.5, com outros métodos disponíveis na literatura.

A primeira comparação considerou quatro abordagens de algoritmos genéticos: algoritmos AG_GNE_LIT e AG_GGA_LIT propostos por Pinto (PINTO, 2014) e algoritmos GNE_LIT e GGA_LIT propostos por Praditwong (PRADITWONG, 2011). Em relação ao algoritmo genético AG_GNE_LIT é possível concluir que, para instâncias grandes, o BRKGA_PCMS_BL é mais eficiente, já que obteve a melhor média de MQ em todas as instâncias grandes. Em relação ao AG_GGA_LIT, o BRKGA_PCMS_BL obteve médias de MQ inferiores na maioria das instâncias pequenas, médias e grandes (somente em seis das 45 instâncias o BRKGA_PCMS_BL obteve médias melhores), sendo assim, a estratégia de Pinto (PINTO, 2014), baseada em agrupamentos com a utilização de operadores de recombinação e mutação atuando sobre os *clusters* em vez dos módulos, mostrou-se mais eficiente do que a estratégia utilizada pelo BRKGA_PCMS_BL. Em relação ao tempo de execução, o BRKGA_PCMS_BL é mais lento que ambas heurísticas (aproximadamente 4,6 vezes mais lento que o AG_GNE_LIT e aproximadamente 9,32% mais lento que o AG_GGA_LIT). Em relação aos genéticos GNE_LIT e GGA_LIT, o BRKGA_PCMS_BL mostrou-se eficaz na resolução do PCMS (não foi possível realizar uma comparação dos tempos de execução pois estes não foram reportados em Praditwong (PRADITWONG, 2011)). Na comparação com o GNE_LIT, o BRKGA_PCMS_BL obteve melhores médias de MQ em 87,5% dos casos (14 das 16 instâncias testadas) e na comparação com o GGA_LIT o BRKGA_PCMS_BL obteve melhores médias de MQ em 75% dos casos (12 das 16 instâncias), sendo que as quatro instâncias em que o BRKGA_PCMS_BL foi derrotado eram instâncias pequenas.

A próxima comparação considerou a heurística ILS_CMS, proposta por Pinto (PINTO, 2014), baseada na metaheurística ILS. A ILS_CMS obteve a melhor média de MQ em 80% dos casos. Em termos de eficiência, o ILS_CMS obteve excelentes resultados em comparação ao tempo de execução do BRKGA_PCMS_BL (seu custo computacional é aproximadamente 80% inferior). O mesmo resultado em termos de eficiência já havia sido verificado por Pinto (PINTO, 2014) quando comparando seu método com outros genéticos da literatura.

Na comparação com o método exato $\text{MILP}^{+}_{2,1}$, proposto por Köhler (KÖHLER; FAMPA; ARAUJO, 2013), é possível observar que o BRKGA_PCMS_BL obteve o mesmo resultado de MQ para cinco instâncias enquanto que o $\text{MILP}^{+}_{2,1}$ obteve melhores resultado de MQ para 11 instâncias. Apesar da aparente superioridade do método exato não é possível afirmar que este é mais eficiente pois a média de melhoria de MQ (*gap*) ficou em aproximadamente 0,6% a um custo computacional muito elevado, aproximadamente 627 vezes mais lento que o BRKGA_PCMS_BL.

Finalmente, na comparação com os melhores resultados conhecidos de 113 instâncias da literatura, reportados na Seção 4.5.4, o BRKGA_PCMS_BL obteve melhores resultados de MQ em dez das 113 instâncias estudadas, sendo que em oito instâncias ocorreram empates.

Para finalizar esta seção, ao analisar todos os experimentos realizados na Seção 4.5, não foi possível concluir que o BRKGA_PCMS_BL seja uma heurística eficiente, nem eficaz, para a resolução do PCMS comparativamente a outras heurísticas frequentemente usadas na resolução deste problema. Entretanto, um dos objetivos deste trabalho era o de realizar um experimento computacional robusto. Acredita-se que cada um dos experimentos realizados, avaliando os pontos fortes e fracos da heurística BRKGA_PCMS_BL, tenha apresentado contribuições que possam servir de referência para trabalhos futuros. Até o presente momento, é desconhecida literatura que tenha usado o BRKGA para resolver o PCMS e, mesmo, literatura que tenha documentado a comparação com um número tão grande de instâncias (113 instâncias) para este problema e comparado com tantos outros métodos (quatro genéticos, um método exato e um ILS).

Por último, outra contribuição desta dissertação é a implementação do *framework* BRKGA utilizando a linguagem de programação `C#.NET`, disponibilizado no site *CodePlex* em https://brkganet.codeplex.com/. Este *framework* pode ser utilizado para qualquer problema que tenha como solução uma permutação de valores sendo que o usuário teria que implementar apenas os codificadores e os decodificadores do problema em questão. Até o momento é desconhecida a implementação do BRKGA nesta tecnologia.

5.2 Limitações e perspectivas de trabalhos futuros

Nesta seção destacam-se limitações do presente trabalho e apresentam-se perspectivas de trabalhos futuros.

Os algoritmos propostos para criar uma solução usam uma estratégia em que um número m de *clusters* é inicialmente aberto e, em seguida, os módulos são inseridos nestes m *clusters*, podendo eventualmente o algoritmo abrir um novo *cluster*. Esta estratégia foi implementada considerando a quantidade de *clusters* da solução pertencente a um intervalo entre 20 e 40% do número n de módulos, calculado com base em dados da literatura. Entretanto, este intervalo é muito restritivo e pode estar impedindo a busca de navegar por outras regiões promissoras. Deseja-se, em trabalhos futuros, estudar e implementar uma estratégia do tipo adaptativa que, por exemplo, inicialmente, considere para a quantidade de *clusters* m da solução qualquer número entre 1 e n. Em seguida, a cada iteração, avalia-se a qualidade das soluções e aumenta-se a probabilidade de sorteio dos valores de m que conseguiram as melhores soluções.

Uma sugestão de aprimoramento da heurística proposta, em termos de eficiência, é a utilização de técnicas de redução de grafos, assim como proposto por Köhler et al. (KÖHLER; FAMPA; ARAUJO, 2013). Essa técnica é uma forma de pré-processamento capaz de reduzir a quantidade de módulos e de dependências nos grafos MDG. Com a redução no tamanho do grafo MDG, o número total de clusterizações possíveis também é reduzido e, com isso, o problema pode ser resolvido com menos esforço computacional. O método de redução criado por Köhler

et al. (KÖHLER; FAMPA; ARAUJO, 2013) para o PCMS garante que, dado um grafo MDG, o valor ótimo do MQ desse MDG é igual ao valor ótimo do MQ do MDG reduzido. Os resultados da redução mostram que, dependendo da instância, a redução pode chegar a aproximadamente 30% do tamanho original do grafo.

Outra sugestão, ainda em termos de eficiência, é a utilização de paralelismo em algumas partes do *framework* BRKGA. Segundo Resende et al. (GONÇALVES; RESENDE, 2011) algumas operações do *framework* que possuem uma natureza independente de execução podem beneficiar-se de execução paralela, incluindo-se: a geração dos vetores de chaves aleatórias, a geração dos mutantes a serem introduzidos na população, a combinação dos pais elite e não elite para gerar as proles, a decodificação de cada chave aleatória e, por último, o cálculo de sua aptidão.

Outra aplicação do BRKGA possível de ser testada em trabalhos futuros é a estratégia utilizada em Gonçalves e Resende (GONÇALVES; RESENDE, 2011) na qual múltiplas populações são utilizadas. A estratégia de múltiplas populações consiste em evoluir, de forma independente, um conjunto de populações. Após um determinado número de gerações, as informações de cromossomos de alta qualidade são trocadas entre si. Existem diversas maneiras de trocar estas informações. No entanto, os autores empiricamente decidiram por copiar os dois melhores cromossomos de todos os cromossomos existentes para uma população onde eles não existem, removendo os dois piores cromossomos desta população.

Outra estratégia a ser investigada refere-se à geração da população inicial. Ao invés de gerar todos os cromossomos aleatoriamente, Gonçalves e Resende (GONÇALVES; RESENDE, 2011) introduzem quatro cromossomos definidos de forma não aleatória. Mais uma ideia que pode ser aproveitada é a utilização de múltiplos pais no processo de geração de uma prole como pode ser visto em Lucena et al. (LUCENA et al., 2014).

Por último, é desejado a utilização de experimentos com gráficos TTTPLOTs (AIEX; RESENDE; RIBEIRO, 2007). Estes gráficos são utilizados na comparação de diferentes algoritmos ou estratégias para um determinado problema onde no eixo y é apresentada a probabilidade de um algoritmo encontrar uma solução tão boa quanto um valor alvo definido em um determinado intervalo de tempo, este, definido no eixo x.

Referências

AIEX, R. M.; RESENDE, M. G.; RIBEIRO, C. C. “TTT plots: a perl program to create time-to-target plots”. *Optimization Letters*, v. 1, n. 4, p. 355–366, 2007.

BARR, R. S. et al. “Designing and reporting on computational experiments with heuristic methods”. *Journal of Heuristics*, v. 1, n. 1, p. 9–32, 1995.

BARROS, M. de O. “An Analysis of the Effects of Composite Objectives in Multiobjective Software Module Clustering”. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2012. (GECCO ’12), p. 1205–1212.

BARROS, M. de O. “An experimental evaluation of the importance of randomness in hill climbing searches applied to software engineering problems”. *Empirical Software Engineering*, Springer US, v. 19, n. 5, p. 1423–1465, 2014.

BAVOTA, G. et al. “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*. Berlin, Heidelberg: Springer-Verlag, 2012. (SSBSE’12), p. 75–89.

BEAN, J. C. “Genetic Algorithms and Random Keys for Sequencing and Optimization”. *ORSA Journal on Computing*, v. 6, n. 2, p. 154–160, 1994.

BOTELHO, A. L. V.; SEMAAN, G. S.; OCHI, L. S. “Agrupamento de Sistemas Orientados a Objetos com Metaheurísticas Evolutivas”. In: *Anais do VII Simpósio Brasileiro de Sistemas de Informação*. Salvador, Bahia, Brasil: [s.n.], 2011.

BRESLAU, L. et al. “Disjoint-Path Facility Location: Theory and Practice”. In: *2011 Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 2011. p. 60–74.

BROOKS JR., F. P. “No Silver Bullet Essence and Accidents of Software Engineering”. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 20, n. 4, p. 10–19, abr. 1987.

CAMI, K.; SRINIVASULU, A. “Multi-Objective Approach for Software Module Clustering”. *International journal of advanced Research in Computer Science and Software Engineering*, v. 2, n. 3, 2012.

CAN, K.; SRINIVASULU, A. “Multi-Objective Approach for Software Module Clustering”. *International journal of advanced Research in Computer Science and Software Engineering*, v. 2, n. 3, 2012.

CHEN, D. Z. et al. “Efficient Algorithms and Implementations for Optimizing the Sum of Linear Fractional Functions, with Applications”. *Journal of Combinatorial Optimization*, v. 9, n. 1, p. 69–90, 2005.

CLEVELAND, G. A.; SMITH, S. F. “Using Genetic Algorithms to Schedule Flow Shop Releases”. In: *Proceedings of the Third International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. p. 160–169.

CONSTANTINE; YOURDON. *Structured Design*. [S.l.]: Prentice Hall, 1979.

COWLING, P.; KENDALL, G.; SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In: BURKE, E.; ERBEN, W. (Ed.). *Practice and Theory of Automated Timetabling III*. [S.l.]: Springer Berlin Heidelberg, 2001, (Lecture Notes in Computer Science, v. 2079). p. 176–190. ISBN 978-3-540-42421-5.

DIAS, C. R.; OCHI, L. S. Efficient evolutionary algorithms for the clustering problem in directed graphs. In: SARKER, R. et al. (Ed.). *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*. Canberra: IEEE Press, 2003. p. 983–990.

DOVAL, D.; MANCORIDIS, S.; MITCHELL, B. "Automatic clustering of software systems using a genetic algorithm". In: *Proceedings of the Software Technology and Engineering Practice*. Washington, DC, USA: IEEE Computer Society, 1999. (STEP '99), p. 73–81.

FALKENAUER, E. "A hybrid grouping genetic algorithm for bin packing". *Journal of Heuristics*, Kluwer Academic Publishers, v. 2, n. 1, p. 5–30, 1996.

FELTOVICH, N. "Nonparametric Tests of Differences in Medians: Comparison of the Wilcoxon–Mann–Whitney and Robust Rank-Order Tests". *Experimental Economics*, Kluwer Academic Publishers, v. 6, n. 3, p. 273–297, 2003.

FEO, T. A.; RESENDE, M. G. C.; SMITH, S. H. "A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set". *Operations Research*, v. 42, n. 5, p. 860–878, 1994.

FERREIRA, C. et al. "The node capacitated graph partitioning problem: A computational study". *Mathematical Programming*, v. 81, n. 2, p. 229–256, 1998.

FOGEL, D. B. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ, USA: IEEE Press, 1995. ISBN 0-7803-1038-1.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN 0716710447.

GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook of metaheuristics*. Boston, Dordrecht, London: Kluwer Academic Publishers, 2003. (International series in operations research & management science). ISBN 1-4020-7263-5.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

GOLDBERG, D. E.; LINGLE, R. "Alleles, loci, and the traveling salesman problem". In: LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS. *Proceedings of the first international conference on genetic algorithms and their applications*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985. p. 154–159.

GONÇALVES, J. F.; RESENDE, M. G. "Biased random-key genetic algorithms for combinatorial optimization". *Journal of Heuristics*, Springer US, v. 17, n. 5, p. 487–525, 2011.

GONÇALVES, J. F.; RESENDE, M. G. "Random Key Genetic Algorithms". Aceito para publicação: Handbook of Heuristics. 2014.

GONÇALVES, J. F.; RESENDE, M. G. C. "A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem". *Journal of Combinatorial Optimization*, v. 22, n. 2, p. 180–201, 2011.

GONÇALVES, J. F.; RESENDE, M. G. C.; TOSO, R. F. "An experimental comparison of biased and unbiased random-key genetic algorithms". *Pesquisa Operacional*, scielo, v. 34, p. 143–164, 2014.

GREFENSTETTE, J. J. "Incorporating problem specific knowledge into genetic algorithms". In: DAVIS, L. (Ed.). *Genetic algorithms and simulated annealing*. CA, USA: Morgan Kaufman, 1987. p. 42–60.

GREFENSTETTE, J. J. et al. "Genetic Algorithms for the Traveling Salesman Problem". In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985. p. 160–168.

HANSEN, P.; JAUMARD, B. "Cluster analysis and mathematical programming". *Mathematical Programming*, Springer-Verlag, v. 79, n. 1-3, p. 191–215, 1997.

HARMAN, M. "An empirical study of the robustness of two module clustering functions". In: *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2005)*. Washington DC, USA: Press, 2005. p. 1029–1045.

HARVEY, I. *Evolutionary robotics and SAGA: the case for hill crawling and tournament selection*. Relatório Técnico, School of Cognitive and Computing Sciences, University of Sussex, Brighton, Uk, 1992.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0-262-58111-6.

JONG, K. D. *An Analysis of the Behavior of a Class of Genetic Adaptative Systems*. Tese (Doutorado), 1975.

KNUTH, D. E. *The Art of Computer Programming, volume 2: Seminumerical Algorithms*. 2th. ed. [S.l.]: Addison-Wesley, 1981.

KÖHLER, V.; FAMPA, M.; ARAUJO, O. "Mixed-Integer Linear Programming Formulations for the Software Clustering Problem". *Computational Optimization and Applications*, Springer US, v. 55, n. 1, p. 113–135, 2013.

KRAMER, H. H. et al. "Column generation approaches for the software clustering problem". In: *XLVI Simpósio Brasileiro de Pesquisa Operacional*. Salvador: [s.n.], 2014. p. 2639–2650.

KUMAR, R. "Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms". *International Journal of Machine Learning and Computing*, 2012.

KUMARI, A. C.; SRINIVAS, K.; GUPTA, M. P. "Software module clustering using a hyper-heuristic based multi-objective genetic algorithm". In: *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. Ghaziabad, India: IEEE, 2013. p. 813–818.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). *Handbook of Metaheuristics*. [S.l.]: Springer US, 2003, (International Series in Operations Research & Management Science, v. 57). p. 320–353.

LUCENA, M. L. et al. "Some extensions of biased random-key genetic algorithms". In: *XLVI Simpósio Brasileiro de Pesquisa Operacional*. Salvador: [s.n.], 2014. p. 2469–2480.

MAHDAVI, K.; HARMAN, M.; HIERONS, R. M. A multiple hill climbing approach to software module clustering. In: *Proceedings of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2003. (ICSM '03), p. 315–.

MAMAGHANI, A.; MEYBODI, M. "Clustering of Software Systems Using New Hybrid Algorithms". In: *Ninth IEEE International Conference on Computer and Information Technology, CIT, 2009*. Xiamen, China: IEEE, 2009. v. 1, p. 20–25.

MANCORIDIS, S. et al. "Using automatic clustering to produce high-level system organizations of source code". In: *Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on*. Ischia: IEEE, 1998. p. 45–52.

MANCORIDIS, S. et al. "Bunch: a clustering tool for the recovery and maintenance of software system structures". In: *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM '99)*. Oxford, England: IEEE, 1999. p. 50–59.

MITCHELL, B. "A heuristic search approach to solving the software clustering problem". In: *Proceedings International Conference on Software Maintenance, 2003. ICSM 2003*. Amsterdam, The Netherlands: IEEE, 2003. p. 285–288.

MITCHELL, B. S. *A heuristic search approach to solving the software clustering problem*. Tese (Doutorado) — Drexel University, Philadelphia, PA, USA, 2002.

MITCHELL, M. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996. ISBN 0-262-13316-4.

MORÁN-MIRABAL, L. et al. "Randomized heuristics for handover minimization in mobility networks". *Journal of Heuristics*, v. 19, n. 6, p. 845–880, 2013.

PINTO, A. F. *Uma heurística baseada em buscal local iterada para o problema de clusterização de módulos de software*. Dissertação (Mestrado) — PPGI/UNIRIO, Rio de Janeiro, RJ, Brasil, 2014.

PINTO, A. F.; ALVIM, A. C. de F.; BARROS, M. de O. "ILS for the Software Module Clustering Problem". In: *XLVI Simpósio Brasileiro de Pesquisa Operacional*. Salvador: [s.n.], 2014. p. 1972–1983.

POLI, R.; LANGDON, W. "Genetic Programming with One-Point Crossover". In: CHAWDHRY, P.; ROY, R.; PANT, R. (Ed.). *Soft Computing in Engineering Design and Manufacturing*. [S.l.]: Springer London, 1998. p. 180–189.

PRADITWONG, K. "Solving software module clustering problem by evolutionary algorithms". In: *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*. Nakhon Pathom: IEEE, 2011. p. 154–159.

PRADITWONG, K.; HARMAN, M.; YAO, X. "Software Module Clustering as a Multi-Objective Search Problem". *IEEE Transactions on Software Engineering*, v. 37, n. 2, p. 264–282, March 2011.

PRADITWONG, K.; YAO, X. "A New Multi-objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm". In: *2006 International Conference on Computational Intelligence and Security*. Guangzhou: IEEE, 2006. v. 1, p. 286–291.

QUIROZ-CASTELLANOS, M. et al. “A grouping genetic algorithm with controlled gene transmission for the bin packing problem”. *Computers & Operations Research*, v. 55, n. 0, p. 52 – 64, 2015.

REEVES, C. R. “Genetic Algorithms for the Operation Researcher”. *INFORMS Journal on Computing*, v. 9, n. 3, p. 231–250, 1997.

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 (SPOLM2007)*. Rio de Janeiro, Brasil: [s.n.], 2007. (SPOLM 2007).

SÍMA, J.; SCHAEFFER, S. E. “On the NP-Completeness of Some Graph Cluster Measures”. In: *Proceedings of the 32Nd Conference on Current Trends in Theory and Practice of Computer Science*. Berlin, Heidelberg: Springer-Verlag, 2006. (SOFSEM’06), p. 530–537.

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, p. 798–816, nov. 2010.

SPEARS, V. M.; JONG, K. A. D. “On the virtues of parameterized uniform crossover”. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Diego, CA, USA: Morgan Kaufmann, 1991. p. 230–236.

SRINIVAS, N.; DEB, K. “Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms”. *Evol. Comput.*, Cambridge, MA, USA, v. 2, n. 3, p. 221–248, set. 1994.

STEFANELLO, F. et al. “A Biased Random-key Genetic Algorithm for Placement of Virtual Machines Across Geo-Separated Data Centers”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2015. (GECCO ’15), p. 919–926.

TAILLARD, É. D. “Comparison of Non-Deterministic Iterative Methods”. In: *Metaheuristic Interantional Conference (MIC’01) proceedings*. Porto, Portugal: [s.n.], 2001. p. 273–276.

THE R Project for Statistical Computing. 2015. [Último acesso em 30 de setembro de 2015]. Disponível em: <https://www.r-project.org/>.

TUCKER, A.; SWIFT, S.; LIU, X. “Variable grouping in multivariate time series via correlation”. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, v. 31, n. 2, p. 235–245, Apr 2001.

VANDERBECK, F. “Branching in branch-and-price: a generic scheme”. *Mathematical Programming*, v. 130, n. 2, p. 249–294, 2011.

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*, Sage Publications, v. 25, n. 2, p. 101–132, 2000.

WHITLEY, D. “A Genetic Algorithm Tutorial”. *Statistics and Computing*, Kluwer Academic Publishers, v. 4, n. 2, p. 65–85, 1994.