UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Algoritmos Baseados em Estimadores de Distribuição e Técnicas da Otimização com Muitos Objetivos aplicados na Refatoração Automática de Software

Dissertação de Mestrado

Glauber Andrade Botelho

São Cristóvão – Sergipe

2017

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Glauber Andrade Botelho

Algoritmos Baseados em Estimadores de Distribuição e Técnicas da Otimização com Muitos Objetivos aplicados na Refatoração Automática de Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador: Prof. Dr. André Britto de Carvalho
Coorientadora: Prof. Dra. Leila Maciel de Almeida e Silva

São Cristóvão – Sergipe

2017

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL UNIVERSIDADE FEDERAL DE SERGIPE

Botelho, Glauber Andrade
B748a Algoritmos baseados em estimadores de distribuição e técnicas da otimização com muitos objetivos aplicados na refatoração automática de software / Glauber Andrade Botelho ; orientador André Britto de Carvalho. – São Cristóvão, 2017.
76 f. : il.

Dissertação (mestrado em Ciências da computação)– Universidade Federal de Sergipe, 2017.

1. Programas de computador. 2. Engenharia de software. 3. Software – Refatoração. I. Carvalho, André Britto de, orient. II. Título.

CDU: 004.416.6

Glauber Andrade Botelho

Algoritmos Baseados em Estimadores de Distribuição e Técnicas da Otimização com Muitos Objetivos aplicados na Refatoração Automática de Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Trabalho aprovado. São Cristóvão – Sergipe, 30 de Agosto de 2017:

Prof. Dr. André Britto de Carvalho
Orientador

Prof. Dra. Leila Maciel de Almeida e Silva
Coorientadora

Prof. Dr. Michel dos Santos Soares
Convidado

Prof. Dr. Ricardo Bastos Cavalcante Prudêncio
Convidado

São Cristóvão – Sergipe
2017

Agradecimentos

Ao professor Dr. André Britto de Carvalho e à professora Dra. Leila Maciel de Almeida e Silva , pela ótima orientação e por se fazerem presentes durante todo o desenvolvimento deste trabalho.

Aos meus colegas de trabalho, por todo o apoio prestado perante as dificuldades encontradas durante esta jornada.

Aos meus familiares e amigos, por todo o incentivo dado e pela confiança que sempre depositaram em mim.

Resumo

A Engenharia de Software Baseada em Busca, conhecida como SBSE (do inglês, *Search Based Software Engineering*), é uma área que usa algoritmos de otimização para solucionar problemas da Engenharia de Software. A área de SBSE compreende diversas subáreas, dentre estas, encontra-se a Refatoração de Software Baseada em Busca (SBSR, do inglês *Search Based Software Refactoring*), que trata de processos de refatoração automática de software utilizando algoritmos de otimização. Um dos problemas encontrados na SBSR é a determinação de sequências de refatorações com o objetivo da melhoria do código, de acordo com critérios previamente determinados. Para modelar este problema como um problema de otimização é necessário que sejam definidas funções objetivo. Nesse caso, as funções objetivo são métricas de qualidade utilizadas na Engenharia de Software. Como são muitas as métricas a se considerar, o problema de determinação de sequências de refatorações é naturalmente um Problema de Otimização combinatório com Muitos Objetivos. A otimização com muitos objetivos compreende um conjunto de algoritmos e técnicas que buscam resolver problemas de otimização com mais de três funções objetivo. Para resolver problemas dessa natureza, são propostos novos métodos que visam reduzir a deterioração da busca à medida que o número de objetivos aumenta. Dentre eles, destacam-se os algoritmos evolucionários multiobjetivo, que incluem os *Estimation of Distribution Algorithms* (EDA). No entanto, apesar de ter bons resultados em problemas de otimização combinatória, os EDA ainda são pouco explorados no contexto da otimização com muitos objetivos. O objetivo deste trabalho é investigar o uso dos EDA na seleção automática de uma sequência de refatorações. Para que esse objetivo seja atingido, propõe-se um novo EDA, no qual são incorporadas técnicas de otimização de muitos objetivos existentes na literatura. O algoritmo proposto foi validado e adicionado a um *framework* de refatoração automática de software previamente implementado. Após a condução de um conjunto de experimentos, os resultados obtidos pelo algoritmo proposto foram comparados aos resultados obtidos pelos algoritmos encontrados na literatura aplicados no contexto de SBSR.

Palavras-chave: Refatoração, Otimização, Engenharia de Software.

Abstract

Search Based Software Engineering (SBSE) is a research area in which optimization algorithms are applied to solve Software Engineering problems. SBSE area comprises several sub-areas, among them, Search Based Software Refactoring (SBSR), which deals with automatic software refactoring processes. One of the problems investigated in SBSR is the determination of a sequence of refactorings that provides code improvement, according to predetermined criteria. To model this problem as an optimization problem, it is necessary to define objective functions. In this case, the objective functions are quality metrics used in Software Engineering. Since there are many metrics to consider, the problem of determining sequences of refactorings is a Many-Objective Combinatorial Optimization Problem. Many-Objective optimization comprises a set of algorithms and techniques used to solve problems that take into account more than three objectives. In order to solve problems of this nature, new methods are proposed to reduce the deterioration of the search as the number of objectives increases, among them, the multi-objective evolutionary algorithms, which include the Estimation of Distribution Algorithms (EDA). However, despite having good results in combinatorial optimization problems the investigation of EDA in the context of optimization with many objectives are still incipient. The goal of this work is to investigate the use of EDA in the automatic selection of a sequence of refactorings. In order to achieve this goal, a new EDA is proposed, which incorporates many-objective optimization techniques existing in literature. The proposed algorithm was validated and added to a previously implemented automatic software refactoring framework. Furthermore, a set of experiments was conducted to evaluate the proposed algorithm. The results obtained by the proposed algorithm were compared to the results obtained by algorithms existing in the literature.

Keywords: Refactoring, Optimization, Software Engineering.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACO	*Ant Colony Optimization*
BMDA	*Bivariate Marginal Distribution Algorithm*
BOA	*Bayesian Optimization Algorithm*
CD	*Crowding Distance*
Code-Imp	*Combinatorial Optimization Design-Improvement*
EDA	*Estimation of Distribution Algorithms*
EHM	*Edge Histogram Matrix*
FDA	*Factorized Distribution Algorithm*
IBEA	*Indicator-Based Evolutionary Algorithm*
MAHC	*Modified Adaptative Hill Climbing*
mBOA	*Multi-Objective Bayesian Optimization Algorithm*
MGA	*Multi-level Grid Archiving*
MOEA	*Multi-Objective Evolutionary Algorithms*
MOP	*Multi-Objective Optimization Problems*
MOPSO	*Multi-Objective Particle Swarm Optimization*
MOPSOPR	*Multi-Objective Particle Swarm Optimization with Path Relinking*
NSGA-II	*Elitist Non-dominated Sorting Genetic Algorithm II*
NSGA-III	*Elitist Non-dominated Sorting Genetic Algorithm III*
OO	Orientado a Objetos
PSO	*Particle Swarm Optimization*
QMOOD	*Quality Model for Object-Oriented Design*
SBSE	*Search Based Software Engineering*
SBSR	*Search Based Software Refactoring*
SC	Software Controle

SMPSO *Speed-constrained Multi-objective Particle Swarm Optimization*

SMS-EMOA *S Metric Selection Evolutionary Multiobjective Optimization Algorithm*

SPEA2 *Strength Pareto Evolutionary Algorithm 2*

TSP *Traveling Salesman Problem*

UMDA *Univariate Marginal Distribution Algorithm*

Sumário

1 Introdução

Nos últimos anos, tem crescido o interesse no estudo de técnicas de Engenharia de Software Baseada em Busca (SBSE, do inglês *Search Based Software Engineering*), uma abordagem na qual conceitos de otimização são aplicados na resolução de problemas da Engenharia de Software. Técnicas da SBSE são aplicadas na resolução de problemas presentes ao longo do ciclo de vida do software, desde o levantamento de requisitos até a fase de manutenção. A abordagem é atrativa por ser capaz de automatizar, total ou parcialmente, a resolução de problemas que possuem um espaço de busca das soluções extenso e lidam com objetivos conflitantes entre si. (HARMAN, 2006)

Técnicas de SBSE são aplicadas nas várias fases do ciclo de vida do software, dentre os subdomínios da SBSE, pode-se destacar a Refatoração de Software Baseada em Busca (SBSR, do inglês *Search Based Software Refactoring*)(MARIANI; VERGILIO, 2017). Na SBSR, a busca por sequências de refatorações de software é mapeada como um problema de otimização. Os esforços realizados para desenvolver a pesquisa nessa área são motivados pela necessidade de automatizar o procedimento de refatoração de software, uma vez que espera-se uma melhoria nos processos de refatoração se o montante de dados envolvidos puder ser analisado de forma automática ou semi-automática.

Para que a busca por sequências de refatorações seja mapeada como um problema de otimização, é necessário definir um critério segundo o qual seja possível comparar duas sequências de refatorações e decidir qual delas é melhor; para isso, são utilizadas métricas definidas na Engenharia de Software. No processo de mapeamento do problema como um problema de otimização, é necessário que uma sequência de refatorações seja representada computacionalmente, geralmente esta sequência é representada por um vetor. Além disso, cada métrica é mapeada como uma função objetivo. Visto que cada métrica considera um aspecto diferente do software e que diversas métricas são utilizadas, a busca por sequências de refatorações pode ser classificada como um problema de otimização combinatório com muitos

objetivos.

Enquanto a otimização multiobjetivo busca resolver problemas que possuam dois ou três objetivos, a otimização com muitos objetivos lida com problemas onde são considerados quatro objetivos ou mais (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Em problemas com muitos objetivos, o desempenho dos algoritmos evolucionários tradicionais – como o NSGA-II (DEB et al., 2002a) e o PSO (KENNEDY; EBERHART, 1995) – piora à medida que o número de funções objetivo cresce. Várias abordagens têm sido propostas objetivando a adaptação dos algoritmos evolucionários multiobjetivo a problemas com muitos objetivos. Dentre estas abordagens, estão a modificação da definição tradicional da dominância de Pareto (SATO; AGUIRRE; TANAKA, 2007), a combinação de critérios baseados na dominância de Pareto com outras métricas relacionadas à convergência (KÖPPEN; YOSHIDA, 2007) e o desenvolvimento de novos critérios de seleção baseados em indicadores de performance, como o IBEA (ZITZLER; KÜNZLI, 2004) e o SMS-EMOA (EMMERICH; BEUME; NAUJOKS, 2005). Para que técnicas de otimização com muitos objetivos sejam exploradas é necessário que elas sejam implementadas em conjunto com um algoritmo projetado para resolver problemas de otimização. Nesse contexto, encontra-se o *Estimation of Distribution Algorithm* (EDA). Os EDA são meta-heurísticas populacionais que se baseiam em um modelo probabilístico para realizar a busca por soluções. Em trabalhos anteriores, os EDA já foram explorados em problemas combinatórios e no contexto de SBSE ((STAUNTON; CLARK, 2011a),(STAUNTON; CLARK, 2011b)).

A pesquisa na área de otimização com muitos objetivos é direcionada para o desenvolvimento de métodos através dos quais algoritmos possam ser escalados de maneira que a qualidade da busca por soluções se mantenha à medida que o número de objetivos do problema aumenta. Na literatura são encontrados alguns trabalhos onde é investigada a aplicação de algoritmos de busca na refatoração automática de software considerando até 3 objetivos, dentre eles estão (O'KEEFFE; CINNÉIDE, 2006), (O'KEEFFE; CINNÉIDE, 2007), (O'KEEFFE; CINNÉIDE, 2008a) e (O'KEEFFE; CINNÉIDE, 2008b).

O trabalho apresentado em (BEZERRA, 2014) propõe um *framework open-source* que possibilita a busca e a aplicação automática de sequências de refatorações. Este *framework* foi construído utilizando a estrutura do jMetal (DURILLO; NEBRO, 2011), uma biblioteca que permite a execução de diferentes meta-heurísticas. Além do *framework*, o autor propõe o *Multi-Objective Particle Swarm Optimization with Path Relinking* (MOPSOPR), algoritmo desenvolvido para ser utilizado em problemas de SBSR.

Apesar dos resultados obtidos nesses trabalhos, ainda existem limitações que podem ser melhor trabalhadas. Em (O'KEEFFE; CINNÉIDE, 2006), (O'KEEFFE; CINNÉIDE, 2007), (O'KEEFFE; CINNÉIDE, 2008a) e (O'KEEFFE; CINNÉIDE, 2008b) são considerados até três objetivos e o problema de busca por sequência de refatorações não é explorado como um problema de otimização com muitos objetivos nestes trabalhos. O trabalho apresentado em (BEZERRA, 2014) é focado no desenvolvimento do *framework* e não na exploração de técnicas

de otimização com muitos objetivos.

Pode-se perceber, através dos trabalhos anteriores, que uma das lacunas dos trabalhos anteriores é a não exploração de técnicas de otimização com muitos objetivos. Apesar da natureza de problemas com muitos objetivos, poucos trabalhos de fato exploram técnicas de otimização com muitos objetivos na SBSR. Há ferramentas, e alguns trabalhos que avaliam alguns algoritmos, porém, esse conjunto de algoritmos ainda é limitado ((O'KEEFFE; CINNÉIDE, 2006), (O'KEEFFE; CINNÉIDE, 2007), (O'KEEFFE; CINNÉIDE, 2008a), (O'KEEFFE; CINNÉIDE, 2008b)).

1.1 Hipótese

Algoritmos baseados em estimadores de distribuição se mostram uma abordagem para resolver problemas combinatórios e, até então, a aplicação de técnicas de otimização com muitos objetivos foi pouco explorada no contexto de SBSE. A hipótese supõe que a aplicação de um EDA em conjunto com técnicas de otimização com muitos objetivos no problema de busca por sequências de refatorações de software pode trazer bons resultados.

1.2 Objetivos

O objetivo deste trabalho é investigar a utilização de EDA em conjunto com técnicas de otimização com muitos objetivos na busca por sequências de refatorações de software. Apesar de ainda pouco explorados em problemas de otimização multiobjetivo, os EDA têm mostrado bons resultados quando utilizados em problemas combinatórios.

Para alcançar este objetivo, objetivos específicos foram definidos:

- Realizar revisão bibliográfica sobre SBSR e aplicação de EDA em problemas de otimização com muitos objetivos;
- Desenvolver um EDA voltado para a resolução de problemas com muitos objetivos;
- Adaptar o algoritmo proposto para aplicá-lo ao problema da busca por sequências de refatorações;
- Investigar o desempenho do algoritmo proposto no contexto de SBSR.

1.3 Metodologia

Na fase inicial do trabalho foi realizada uma revisão bibliográfica referente a algoritmos de otimização com muitos objetivos e a SBSR. A partir desta revisão foi possível estabelecer a fundamentação teórica do trabalho. Uma vez realizada a revisão bibliográfica, foram definidas

fases a serem cumpridas para que o objetivo do trabalho fosse alcançado. A primeira etapa consistiu na implementação de um EDA básico a ser utilizado para resolver um problema de *benchmarking* para observar, de maneira geral, o comportamento do EDA quando aplicado em um problema com muitos objetivos. Na segunda etapa, com o objetivo de melhorar o desempenho do EDA implementado em problemas de otimização com muitos objetivos, foi investigada a utilização de métodos de arquivamento em conjunto com o EDA. Por fim, o EDA foi adaptado para resolver o problema da busca por sequências de refatorações de software. O processo de adaptação do EDA foi crucial, pois as características do problema de *benchmarking* e do problema de busca por refatorações são diferentes, diante disso, o estimador utilizado pelo EDA precisou ser modificado. Para avaliar o desempenho do EDA, foram executados conjuntos de experimentos visando avaliar os diferentes parâmetros e estimadores utilizados pelo algoritmo, tanto no problema de *benchmarking* quanto no problema de SBSR.

1.4 Organização do trabalho

O restante desta dissertação está organizado da forma descrita a seguir. No Capítulo 2, são apresentados os conceitos de Engenharia de Software Baseada em Busca e de SBSR. Além disso, o problema aqui investigado é definido, os conceitos de otimização com muitos objetivos são introduzidos no Capítulo 3, enquanto o detalhamento sobre o EDA desenvolvido neste trabalho é discutido no Capítulo 4. No Capítulo 5, os experimentos realizados são detalhados. Finalmente, no Capítulo 6, são apresentadas as considerações finais e direções de trabalhos futuros.

2

Refatoração de Software Baseada em Busca

Na Engenharia de Software é comum lidar com problemas que requeiram um balanceamento entre objetivos conflitantes. À medida que o desenvolvimento de software se torna mais complexo, tais problemas envolvem a consideração de inúmeras variáveis e encontrar uma boa solução pode não ser uma tarefa trivial.

Engenharia de Software Baseada em Busca, ou SBSE, é o nome dado ao campo de trabalho onde algoritmos de otimização são aplicados na Engenharia de Software (HARMAN, 2006). O objetivo da SBSE é resolver problemas de Engenharia de Software cujo espaço de busca de solução é amplo, com a aplicação de técnicas de otimização. De forma simplificada, otimização pode ser entendida como uma maneira automática de gerar soluções e caminhar para a melhor solução possível a partir da definição de uma medida quantitativa. Com essa medida é possível comparar de maneira inequívoca duas soluções, sendo possível determinar de maneira não-ambígua qual delas é melhor (TAKAHASHI, 2007). Dentre essas técnicas de otimização, no contexto da SBSE, se destacam aquelas baseadas na aplicação de meta-heurísticas, como os algoritmos genéticos (MITCHELL, 1998), o *Simulated Annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) e a *Ant Colony Optimization* (ACO) (DORIGO; STÜTZLE, 2004).

Na SBSE, um problema de otimização é aquele no qual a busca por soluções ótimas ou quase ótimas é realizada em um conjunto de soluções candidatas. Essa busca é guiada por uma função objetivo que verifica a qualidade da solução. O que torna atrativa a abordagem da SBSE é a necessidade de apenas dois elementos-chave para solucionar o problema: uma representação adequada do problema e uma função objetivo. Geralmente, o engenheiro de software possui uma representação adequada para o problema e tem conhecimento de métricas que podem vir a ser boas candidatas a se tornarem uma função objetivo. Esses fatores tornam a infraestrutura necessária e a curva de aprendizagem de técnicas que usam SBSE baixas e, portanto, relativamente fáceis de serem aplicadas.

A utilização da SBSE pode auxiliar na resolução de problemas encontrados em quase

todas as fases do desenvolvimento de software. Por causa disso, existem subdivisões que lidam com problemas de áreas diferentes. Algumas dessas áreas são:

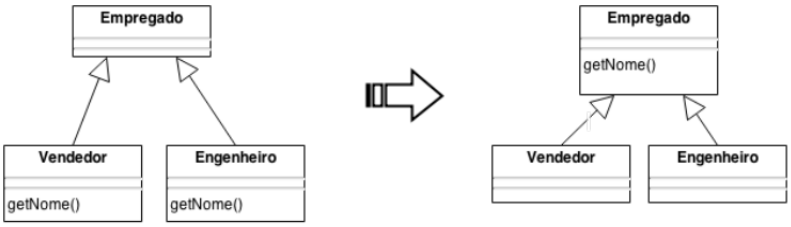
- Engenharia de requisitos: Métodos baseados em SBSE têm sido usados na seleção de requisitos com o objetivo de encontrar o melhor subconjunto possível que satisfaça as requisições feitas por clientes e respeite limitações em termos de recursos (KUMARI; SRINIVAS, 2013), (CHAVES-GONZÁLEZ; PÉREZ-TOLEDANO, 2015), (BOTELHO et al., 2015);
- Otimização de Software: nessa área, técnicas baseadas em SBSE são usadas a fim de conseguir que um determinado software tenha um melhor desempenho em termos de velocidade e utilização de recursos (PRADITWONG, 2011), (ALETI et al., 2013), (LANGDON; HARMAN, 2014);
- *Debug* e manutenção de software: um dos objetivos da SBSE nessa área é identificar e corrigir *bugs* automaticamente (TZOREF; UR; YOM-TOV, 2007), (YOM-TOV et al., 2008), (GOUES et al., 2012);
- Refatoração de software: nessa área, técnicas baseadas em SBSE são utilizadas para automatizar a busca por refatorações de software (O'KEEFFE; CINNÉIDE, 2006), (O'KEEFFE; CINNÉIDE, 2007), (O'KEEFFE; CINNÉIDE, 2008a). A utilização de SBSE na refatoração de software é o foco desta pesquisa, mais detalhes sobre isto são apresentados na próxima seção.

2.1 Refatoração de software

O aumento da importância do software nos processos de negócio evidenciou problemas como a erosão de software, apresentado em (SILVA; BALASUBRAMANIAM, 2012), e o débito técnico (CUNNINGHAM, 1992). O problema da erosão de software é definido como a deterioração de desempenho que o software enfrenta em decorrência de mudanças de ambiente e manutenções inadequadas. Já o problema do débito técnico é referente ao esforço que deve ser feito para corrigir problemas na estrutura geral das aplicações. Para que esses problemas sejam resolvidos, pode-se fazer uso de refatoração. Em (FOWLER, 1999), refatoração é definida como sendo o "processo de alterar o software de uma maneira que o comportamento externo do código fique inalterado, mas que sua estrutura interna melhore". O autor apresenta 72 refatorações que podem ser aplicadas em sistemas orientados a objetos. Um exemplo de uma dessas refatorações é apresentado na Figura 1.

No diagrama apresentado no lado esquerdo da Figura 1, o método *getNome* está presente nas classes *Vendedor* e *Engenheiro*, ambas subclasses da classe *Empregado*. Após a aplicação da refatoração *Pull Up Method*, o método *getNome* é retirado das subclasses e movido para a classe *Empregado*.

Figura 1 – Exemplo de refatoração *Pull Up Method*.



Fonte: Bezerra (2014)

2.2 Refatoração de software como um problema de otimização

Conforme apresentado em (MURPHY-HILL; PARNIN; BLACK, 2009), mais de 90% das refatorações são feitas manualmente. Além disso, os autores mencionam que sem os critérios automáticos de verificação de pré e pós-condições, refatorações executadas manualmente podem gerar erros que podem ser de difícil detecção e tratamento. Assim, é necessário que o processo de aplicação de refatorações seja realizado com cuidado para que os efeitos disso são sejam contrários ao esperado.

Diante disso, mecanismos que possibilitam a aplicação de refatorações automáticas de software aparecem como uma alternativa segura à aplicação de refatorações manuais. Uma das maneiras de trabalhar com refatoração automática de software é utilizando técnicas de Refatoração de Software Baseada em Busca. A área de Refatoração de Software Baseada em Busca, ou SBSR, trata de processos de refatoração automática de software que utilizam algoritmos de otimização. Os trabalhos desta área objetivam encontrar sequências de refatorações que seguem critérios pré-determinados. A aplicação das refatorações selecionadas reflete em melhorias destes critérios previamente definidos.

Para que seja possível utilizar técnicas baseadas em SBSR na busca por sequências de refatorações é necessário que o problema seja mapeado como um problema de otimização. Um problema de otimização é composto por dois elementos: a representação do problema e uma função (ou funções) objetivo utilizada(s) para guiar a busca. Nas seções a seguir estes elementos são detalhados.

2.2.1 Representação do problema de refatoração como um problema de otimização

O problema da busca por sequências de refatorações consiste em encontrar sequências ordenadas de refatorações que melhorem o desempenho do software sem alterar o seu comportamento externo e é definido em (BEZERRA, 2014) da seguinte forma: uma vez que o estado

S_i de um software S pode ser definido pelo seu código-fonte, uma refatoração pode ser descrita como um conjunto de mudanças no código-fonte que modificam este software do estado S_i para um estado S_j. Considerando que a reestruturação do software se dá por meio da aplicação de n refatorações, pode-se dizer que o processo completo de refatoração consiste na aplicação de uma sequência ordenada R de refatorações. Utiliza-se a notação $R_i.S_i$ para representar a aplicação da sequência de refatorações R_i no software S, quando ele está no estado S_i.

Cada refatoração $r \in R_i$ pode ser representada por meio de uma tripla do tipo $r = (I, P, D)$. Onde I é o identificador da refatoração, P representa os parâmetros necessários para que a refatoração seja executada e D é a diferença entre o valor da função objetivo antes da aplicação da refatoração e o valor depois da aplicação da refatoração. Caso sejam consideradas várias funções objetivo, D será um conjunto de valores $D = \{\Delta f_0, \Delta f_1, ..., \Delta f_n\}$, onde Δf_i é a variação da função objetivo f_i após a aplicação da refatoração r.

Por exemplo, uma refatoração pode ser representada pela tripla $(PUF, \{SBC1, SBC2, F1, SPC1\}, +0,2)$, que pode indicar uma refatoração *Pull Up Field*, que move o campo $F1$ das subclasses $SBC1$ e $SBC2$ para a superclasse $SPC1$ e, com essa operação, aumenta o valor da função objetivo em $0,2$.

Conforme apresentado anteriormente, são necessárias funções objetivo para guiar a busca quando se trabalha com problemas de otimização. Para o problema investigado nesse trabalho, métricas definidas na Engenharia de Software são mapeadas como funções objetivo para que se possa medir o quão boa é uma sequência de refatorações. Várias métricas de software podem ser utilizadas na busca por uma solução para o problema em questão, o que faz deste um problema de otimização combinatório com muitos objetivos.

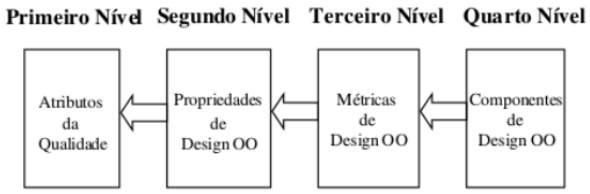
2.2.2 Métricas de software como função objetivo

Para medir o quão melhor ficou um software após a aplicação de uma sequência de refatorações faz-se uso de métricas de software. Alguns exemplos de métricas de software são Linhas de Código, que tem por objetivo indicar o número total de linhas de código da aplicação, e a Complexidade Ciclomática, utilizada para medir o número de caminhos independentes de determinado módulo de um software e, a partir disso, indicar a sua complexidade. Embora ainda utilizadas, Michura, Capretz e Wang (2013) apontam que muitas das métricas de software mais antigas não foram desenvolvidas com foco no paradigma orientado a objetos (OO). As primeiras suítes de métricas que foram desenvolvidas levando em consideração o paradigma OO foram apresentadas em (LORENZ; KIDD, 1994) e (CHIDAMBER; KEMERER, 1991).

Em (BANSIYA; DAVIS, 2002), é apresentado o *Quality Model for Object-Oriented Design* (QMOOD), um modelo hierárquico para análise de atributos de qualidade de software em projetos desenvolvidos à luz do paradigma OO. Nesse modelo, são avaliadas propriedades estruturais e comportamentais de classes, objetos e o relacionamento entre esses elementos. Para

que essa avaliação seja feita, é utilizado um conjunto de métricas utilizadas em projetos OO. Na Figura 2 é apresentado o esquema hierárquico utilizado pelo QMOOD.

Figura 2 – Esquema hierárquico do QMOOD.



Fonte: Bezerra (2014)

No primeiro nível do esquema hierárquico estão os atributos de qualidade de um projeto de software. Inicialmente, os atributos selecionados foram: funcionalidade, confiabilidade, usabilidade, manutenibilidade e portabilidade (atributos definidos pela ISO 9126, na época em que o QMOOD foi proposto). Entretanto, após revisão realizada pelos autores do QMOOD, a lista de atributos foi redefinida para: flexibilidade, funcionalidade, entendimento, efetividade, extensibilidade e reusabilidade. As descrições de cada um desses atributos estão apresentadas na Tabela 1.

Tabela 1 – Definições dos atributos de qualidade do QMOOD.

Atributos de qualidade	Descrição
Flexibilidade	Característica que permite a incorporação de mudanças no projeto de software.
Reusabilidade	Reflete a presença de características que permitem que o projeto de software possa ser reaplicado a um novo problema com um mínimo de esforço.
Entendimento	Avalia se o projeto de software pode ser facilmente entendido e compreendido. Isso está diretamente ligado à complexidade da estrutura.
Funcionalidade	As responsabilidades atribuídas às classes de um projeto de software, disponibilizadas pelas classes através de suas interfaces públicas.
Extensibilidade	Refere-se à presença e utilização de propriedades em um projeto de software existente que permitem a incorporação de novos requisitos no projeto de software atual.
Efetividade	Refere-se à habilidade do projeto de software de alcançar o comportamento e funcionalidade adequadas utilizando técnicas e conceitos de orientação a objetos.

Fonte: Bezerra (2014)

As propriedades de um projeto OO que podem influenciar a qualidade de um projeto são identificadas no segundo nível do esquema hierárquico. O terceiro nível é composto por métricas propostas para avaliar as propriedades que compõem o nível anterior, as descrições das métricas são apresentadas na Tabela 2. A avaliação das características que compõem o segundo nível do modelo é feita por meio de uma soma ponderada das métricas do terceiro nível, os pesos

atribuídos a cada métrica em relação as características são apresentados na Tabela 3. No último nível, encontram-se os elementos que podem ter influência na qualidade de um projeto, dentre esses elementos estão: atributos, métodos, classes, relacionamentos entre classes e hierarquias de classe.

Tabela 2 – Métricas da Suíte QMOOD.

Métrica	Acrônimo	Descrição	Propriedade verificada
Número de classes no projeto (*Design Size in Classes*)	DSC	Número total de classes no projeto.	Tamanho do projeto
Número de hierarquias (*Number of Hierarchies*)	NOH	Quantidade de hierarquias de classes no projeto.	Hierarquias
Média de ancestrais (*Average Number of Ancestors*)	ANA	Representa a média de classes das quais a classe avaliada herda informação. É calculado utilizando o número total de classes na estrutura de herança.	Abstração
Métrica de acesso a dados (*Data Access Metric*)	DAM	Calcula a proporção entre o número de atributos não-públicos e o número total de de atributos declarado em uma classe (faixa 0-1). Calculado como a média entre todas as classes.	Encapsulamento
Acoplamento direto entre classes (*Direct Class Coupling*)	DCC	Uma contagem do número de classes diferentes a que a classe está diretamente relacionada. A métrica inclui classes que são diretamente relacionadas através de declaração de atributos ou passagem de parâmetros em métodos. Calculado como a média entre todas as classes.	Acoplamento
Coesão entre os métodos da classe (*Cohesion Among Methods of class*)	CAM	Representa o grau de relação entre os métodos de uma classe, computado usando a soma da interseção dos parâmetros de um método com o conjunto máximo independente de todos os tipos de parâmetros na classe. Calculado como a média entre todas as classes.	Coesão
Medida de agregação (*Measure Of Aggregation*)	MOA	Uma contagem do número de atributos cujos tipos são classes definidas pelo usuário. Calculado como a média entre todas as classes.	Composição
Medida de abstração funcional (*Measure of Functional Abstraction*)	MFA	A razão entre o número de métodos herdados por uma classe pelo número de métodos acessíveis para aquela classe (faixa 0-1). Calculado como a média entre todas as classes.	Herança
Número de métodos polimórficos (*Number Of Polymorphic methods*)	NOP	Contagem dos métodos que podem exibir comportamento polimórfico. Calculado como a média entre todas as classes.	Polimorfismo
Tamanho da interface de uma classe (*Class Interface Size*)	CIS	Número de métodos públicos em uma classe. Calculado como a média entre todas as classes.	Comunicação
Número de métodos (*Number Of Methods*)	NOM	Número de métodos em uma classe. Calculado como a média entre todas as classes.	Complexidade

Fonte: Bezerra (2014)

Considerando a relevância do modelo, já usado em trabalhos como (O'KEEFFE; CIN-

Tabela 3 – Valores dos pesos das métricas QMOOD para as funções de avaliação.

Elemento avaliado	DSC	NOH	ANA	DAM	DCC	CAM	MOA	MFA	NOP	CIS	NOM
Reusabilidade	0,5	0	0	0	-0,25	0,25	0	0	0	0,5	0
Flexibilidade	0	0	0	0,25	-0,25	0	0,5	0	0.5	0	0
Entendimento	-0,33	0	-0,33	0,33	-0,33	0,33	0	0	-0,33	0	-0,33
Funcionalidade	0,22	0,22	0	0	0	0,12	0	0	0,22	0,22	0
Extensibilidade	0	0	0,5	0	-0,5	0	0	0,5	0,5	0	0
Efetividade	0	0	0,2	0,2	0	0	0,2	0,2	0,2	0	0

Fonte: Bezerra (2014)

NÉIDE, 2006), (O'KEEFFE; CINNÉIDE, 2007), (O'KEEFFE; CINNÉIDE, 2008a), (O'KEEFFE; CINNÉIDE, 2008b) e (BEZERRA, 2014), decidiu-se utilizar o QMOOD também neste trabalho.

2.2.3 Trabalhos relacionados

Na literatura, existem alguns trabalhos relacionados a SBSR, sendo os mais relevantes citados a seguir.

No trabalho apresentado em (SENG; STAMMEL; BURKHART, 2006), os autores utilizam algoritmos genéticos para realizar a busca por sequências de refatorações. O que norteia o processo de busca é a melhoria da coesão, da complexidade e do acoplamento existente entre as classes de um determinado sistema. Estas propriedades são representadas pelo seguinte conjunto de métricas: *Response for Class* (RFC), *Information flow-based-coupling* (ICP), *Tight Class Cohesion* (TCC), *Information flow-based-cohesion* (ICH), *Lack of Cohesion Methods* (LCOM5) e *Weighted Method Count* (WMC). A função objetivo é uma ponderação dos valores de todas essas métricas. No trabalho foram utilizadas as seguintes refatorações: *[Push Down,Pull Up] Field, [Push Down,Pull Up] Method, [Extract, Collapse] Class, Extract Superclass* e *Colapse Class Hierarchy*. Os experimentos realizados mostram uma melhora pouco significativa das métricas após a realização do processo.

Em (O'KEEFFE; CINNÉIDE, 2006), os autores exploram a utilização de algoritmos de busca na automatização do processo de refatoração. Para guiar a busca, são utilizadas as funções de Flexibilidade, Reusabilidade e Entendimento do modelo QMOOD (BANSIYA; DAVIS, 2002). Os autores utilizam três algoritmos: duas variações do *Hill Climbing* e o *Simulated Annealing*. Nos experimentos, são consideradas as seguintes refatorações: *[Push Down,Pull Up] Field, [Push Down,Pull Up] Method* e *[Extract, Collapse] Hierarchy*. O método investigado foi implementado em uma ferramenta proposta pelos autores, o *Code-Imp (Combinatorial Optimization Design-Improvement)*. Os resultados obtidos apontam uma melhora no projeto das aplicações utilizadas nos experimentos, considerando o aumento dos valores das métricas observadas.

Em (O'KEEFFE; CINNÉIDE, 2007), bem como em (O'KEEFFE; CINNÉIDE, 2008b) e (O'KEEFFE; CINNÉIDE, 2008a) os autores apresentam melhorias na ferramenta *Code-Imp*, entre elas, a inclusão de novos pares de refatorações. Além disso, algoritmos genéticos e o

Modified Adaptative Hill Climbing (MAHC) foram acrescentados à lista de algoritmos utilizados. Após realização de experimentos, onde são considerados até três objetivos, verificou-se que o MAHC obteve destaque, enquanto os algoritmos genéticos encontraram soluções de menor qualidade.

No trabalho apresentado em (MKAOUER et al., 2014a), os autores propõem uma abordagem baseada no algoritmo NSGA-III para resolver o problema de busca por sequências de refatorações onde são consideradas 15 métricas de qualidade de software a serem otimizadas. As métricas utilizadas são as seguintes: *Weighted Methods per Class (WMC), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM), Cyclomatic Complexity (CC), Number of Attributes (NA), Attribute Hiding Factor (AH), Method Hiding Factor (MH), Number of Lines of Code (NLC), Coupling Between Object Classes (CBO), Number of Association (NAS), Number of Classes (NC), Depth of Inheritance Tree (DIT), Polymorphism Factor (PF), Attribute Inheritance Factor (AIF)* e *Number of Children (NOC)* (FENTON; PFLEEGER, 1998). Neste trabalho, foram consideradas as refatorações a seguir: *Add Parameter, Rename Method, Encapsulate Collection/ Downcast/Field, Collapse Hierarchy, Hide Method, Extract Class/Interface/Method/Subclass/Superclass, Inline Class/ Method, Move Field/Method, Pull Up Field/Method, Push Down Field/Method* e *Remove Parameter/Setting Method*. A abordagem proposta foi testada em sete sistemas e obteve resultados positivos.

Em Mkaouer et al. (2014b), os autores propõem uma reformulação do problema de busca por sequências de refatorações, baseando-se no fato de que os trabalhos relacionados a este tema não consideram características incertas do projeto, como a criticidade dos trechos de código a serem refatorados e a importância das classes do projeto. Os autores consideram que a criticidade dos trechos de código varia de desenvolvedor para desenvolvedor e que a importância das classes muda a medida que novas classes são adicionadas/removidas do projeto. Diante deste cenário, as funções objetivo consideradas no problema são duas, a qualidade do sistema a ser refatorado e a qualidade das soluções em relação à criticidade dos trechos de código refatorados e à importância das classes do sistema. Os autores propõem uma versão do NSGA-II para resolver o problema, o algoritmo é testado em seis sistemas *open source*. Os resultados evidenciam que a abordagem proposta possibilita a geração de soluções robustas sem uma perda grande de qualidade em cenários variados.

Um *framework* que possibilita a busca e aplicação automática se sequências de refatorações foi proposto em (BEZERRA, 2014), a ferramenta proposta considera as seguintes refatorações: *[Push Down | Pull Up] Field, [Push Down | Pull Up] Method, Encapsulate Field* e *Increase Method Visibility*. Além do *framework*, o autor propõe o algoritmo *Multi-Objective Particle Swarm Optimization with Path Relinking* e investiga a sua aplicação no problema de busca por sequências de refatorações. O algoritmo proposto foi comparado ao NSGA-II, nos experimentos foram consideradas as métricas de qualidade que compõem a suíte QMOOD (BANSIYA; DAVIS, 2002). Apesar de obter alguns resultados positivos, o algoritmo proposto

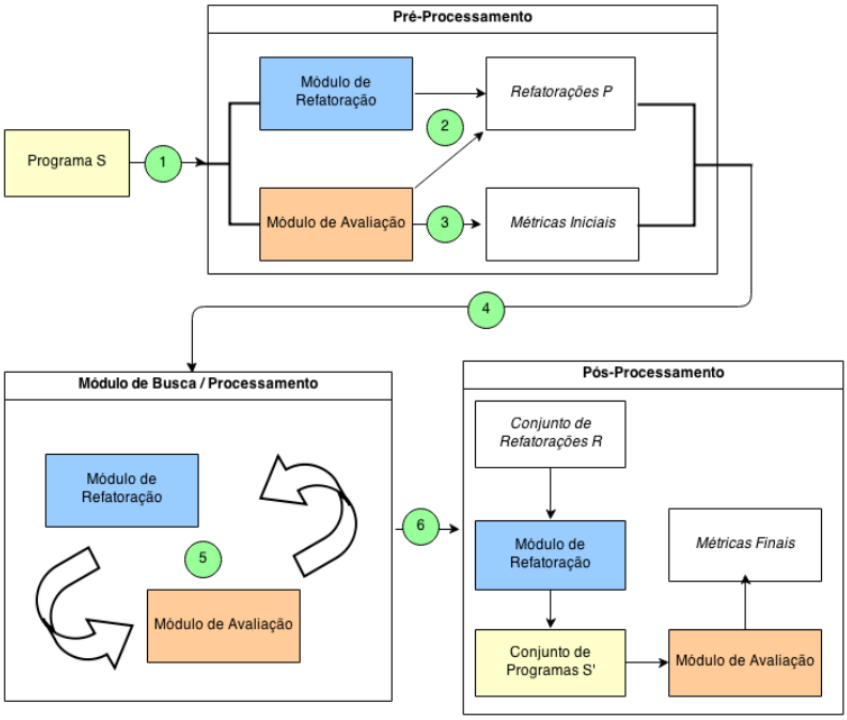
não obteve uma superioridade unânime em relação ao NSGA-II. O *framework* proposto pelo autor foi utilizado neste trabalho e será melhor detalhado na seção a seguir.

2.2.4 O *framework* adotado para refatoração automática de código

Para se trabalhar com o problema de busca por sequências de refatorações são necessárias ferramentas que apoiem e tornem viável a execução dos experimentos realizados durante a pesquisa. Conforme apresentado anteriormente, em (BEZERRA, 2014) o autor propõe um *framework open source* para ser utilizado na busca por sequências de refatorações. O fato do *framework* ser uma ferramenta de código aberto, o que facilita a realização de modificações e adaptações necessárias, e a oportunidade de melhoria e adição de novos algoritmos à ferramenta motivou a utilização do *framework* neste trabalho

As métricas de qualidade que compõem o QMOOD foram utilizadas na ferramenta, com a qual é possível realizar, além da aplicação das refatorações, a avaliação do software resultante. A Figura 3 apresenta uma representação gráfica do *framework*.

Figura 3 – Fluxo basilar do *framework* utilizado.



Fonte: Bezerra (2014)

O *framework* desenvolvido por (BEZERRA, 2014) possui três módulos principais:

- Módulo de refatoração: responsável por gerenciar a aplicação das refatorações e sequências de refatoração, alterando o estado do software para posterior avaliação;

- Módulo de avaliação: avalia o software de acordo com as métricas definidas, o que possibilita a comparação do software em diferentes estados, à medida que as refatorações estão sendo aplicadas;
- Módulo de busca: percorre o espaço de busca, a fim de encontrar sequências de refatorações que melhor otimizem o software.

Conforme elucidado pelo autor, o fluxo de trabalho apresentado na Figura 3 tem início com a seleção de um software *S* que, após ser selecionado, passa por uma etapa de pré-processamento. Em **(1)**, são calculadas todas as métricas do software. Após isso, são descobertas todas as refatorações válidas para o software *S* – as refatorações válidas são aquelas que são aplicadas com sucesso e resultam em alterações na classe – **(2)**. Uma vez que as refatorações válidas tenham sido descobertas, tenta-se aplicá-las ao software avaliado **(3)**. Após as refatorações válidas serem identificadas, o módulo de avaliação recalcula as métricas considerando a aplicação de cada uma dessas refatorações e compara os resultados aos valores obtidos inicialmente. As refatorações válidas são armazenadas em um repositório, juntamente com a variação que a sua aplicação traria aos valores das métricas de qualidade. Esse repositório é utilizado na otimização de operações que necessitam de uma refatoração válida como entrada durante o processo de refatoração automática. Uma vez que o repositório esteja construído, evita-se a busca por essas refatorações sempre que uma operação desse tipo precise ser executada. As refatorações disponíveis no *framework* e suas respectivas descrições são listadas na Tabela 4. Segundo o autor, não foi utilizado um conjunto maior de refatorações devido a limitações encontradas no motor de refatorações utilizado no *framework*.

O próximo passo é a busca pelas melhores sequências de refatorações. Inicialmente, são passados para o módulo de busca o repositório de refatorações válidas previamente construído, as métricas e um conjunto de parâmetros de busca **(4)**. Em **(5)** ocorre a busca pelas sequências de refatorações e, por fim, as sequências de refatorações são reavaliadas e dados estatísticos relacionados ao processo são extraídos **(6)**. No trabalho de Bezerra (2014), no módulo de busca, foi implementada uma versão do algoritmo *Multi-Objective Particle Swarm Optimization with Path Relinking* (MOPSOPR). O algoritmo foi implementado utilizando a estrutura do jMetal (DURILLO; NEBRO, 2011), os resultados foram comparados aos resultados obtidos pelo NSGA-II (DEB et al., 2002b).

Este trabalho utilizará este *framework*, adaptando o módulo de busca, onde será adicionado um novo algoritmo. Os detalhes desta adaptação são apresentados no Capítulo 4.

Tabela 4 – Refatorações utilizadas pelo *framework*.

Refatoração	Descrição
Push Down Field	Move um atributo de uma classe para alguma de suas subclasses. Esta refatoração tenta simplificar o projeto pela redução do número de classes que acessam o atributo.
Pull Up Field	Move um atributo de uma classe para a sua superclasse imediata. Esta refatoração destina-se a eliminar a duplicação de declarações de atributo nas classes irmãs.
Push Down Method	Move um método de uma classe para uma de suas subclasses. Esta refatoração destina-se a simplificar o projeto pela redução de tamanho de interfaces e classes de onde esses métodos deverão sair.
Pull Up Method	Move um método de uma classe para sua superclasse imediata. Esta refatoração destina-se a ajudar a eliminar métodos duplicados entre classes irmãs e, consequentemente, reduzir duplicação de código no geral.
Encapsulate Field	Torna um campo privado e provê acessores. Esta refatoração destina-se a aumentar o encapsulamento, um dos pilares da orientação a objetos.
Increase Method Visibility	Torna um método público. Esta refatoração destina-se a aumentar a interface externa da classe.

Fonte: Bezerra (2014)

2.3 Considerações finais

A SBSR investiga técnicas a serem utilizadas na automatização da busca por sequências de refatorações de software. Considerando o efeito indesejado que a busca e aplicação de sequências de refatorações pode causar, é natural que pesquisas nessa área continuem a ser feitas. Os trabalhos com foco em SBSR publicados até então trouxeram importantes contribuições para o desenvolvimento de técnicas de busca para resolver o problema e para o aprimoramento das métricas de qualidade. Entretanto, a maioria dos trabalhos ainda exploram o tema sem considerar um número grande de objetivos, sendo que a busca por sequências de refatorações se caracteriza naturalmente como um problema de muitos objetivos, uma vez que várias métricas de software podem vir a ser consideradas na sua resolução. Nos próximos capítulos serão apresentados conceitos básicos da otimização com muitos objetivos e o algoritmo proposto neste trabalho.

3

Otimização com Muitos Objetivos

Conforme apresentado no capítulo anterior, o problema de busca por uma sequência de refatorações é um problema onde várias métricas podem ser mapeadas como funções objetivo utilizadas para guiar a busca por soluções. A utilização de um grande número de funções objetivo na resolução de um problema caracteriza um problema de otimização com muitos objetivos. Conceitos e definições referentes à otimização com muitos objetivos são discutidos neste capítulo.

3.1 Otimização multiobjetivo

Segundo Takahashi (2007), a otimização, do ponto de vista prático, consiste em um conjunto de métodos capazes de determinar as melhores configurações possíveis para o funcionamento de sistemas pelos quais o ser humano tem algum interesse. Um problema de otimização é solucionado a partir da exploração do espaço de busca. Soluções são construídas e avaliadas nesse processo, a avaliação das soluções é feita utilizando uma função objetivo, através da qual é possível comparar duas soluções e verificar qual delas é melhor.

Uma instância de um problema de otimização Π é formalmente definida por uma tripla (Ω, f, β), onde Ω é o conjunto de soluções candidatas, f é uma função objetivo que associa um valor $f(s)$ para cada uma das soluções $s \in \Omega$, e β é um conjunto de restrições. As soluções pertencentes ao conjunto $R \subseteq \Omega$ que satisfazem as restrições β são chamadas de soluções possíveis. O objetivo é achar uma solução possível ótima. Para problemas de minimização, isso significa encontrar uma solução $s^* \in \Omega$ que minimize[1] o valor da função objetivo, isto é $f(s^*) \leq f(s)$ para toda solução $s \in \Omega$ (COELLO; LAMONT; VELDHUIZEN, 2006).

Um problema de otimização multiobjetivo lida com mais de uma função objetivo. Devido a falta de mecanismos que encontrassem soluções para os MOP (do inglês, *Multi-Objective Optimization Prob*
no passado eles eram resolvidos como problemas que possuíam apenas um objetivo. Entretanto,

[1] A partir deste ponto, todos os exemplos apresentados consideram problemas de minimização.

há várias diferenças importantes entre algoritmos que resolvem problemas de otimização mono-objetivo e multiobjetivo. Em um problema que possui apenas um objetivo, a tarefa é encontrar uma solução que otimize aquela função objetivo. Nos problemas que possuem mais de uma função objetivo, existem m objetivos a serem otimizados. Pensar que essa otimização deve ser feita para um objetivo de cada vez é errado, os m objetivos devem ser otimizados simultaneamente, dessa maneira, não existe apenas uma melhor solução, existe um conjunto de melhores soluções – obtido através da Teoria da Otimalidade de Pareto (COELLO; LAMONT; VELDHUIZEN, 2006).

Um problema de otimização multiobjetivo é definido por Coello, Lamont e Veldhuizen (2006) da forma descrita a seguir.

Definição 1 (Problema de Otimização Multiobjetivo sem restrições): Seja $\mathbf{x}$ o vetor de variáveis que representa uma possível solução candidata s, $\mathbf{x} = (x_1, x_2, ..., x_n)$. Um problema de otimização multiobjetivo é definido como uma minimização (ou maximização) $F(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_m(\mathbf{x}))$ para $\mathbf{x} \in \Omega$. A solução de um MOP minimiza (ou maximiza) os componentes de um vetor $F(\mathbf{x})$. O universo Ω contém todos os valores que podem ser utilizados em uma avaliação de $F(\mathbf{x})$, onde $F(\mathbf{x})$ é um conjunto de m funções objetivo.

Pelo fato de ter mais de uma função objetivo, a noção de algo ótimo muda em um problema multiobjetivo, porque, em um MOP, o objetivo é encontrar uma solução que seja boa tanto considerando as funções objetivo individualmente, quanto considerando as funções objetivo de uma forma global. Segundo Coello, Lamont e Veldhuizen (2006), o conceito de ótimo mais comumente utilizado foi proposto originalmente por Edgeworth (1881, *apud* Coello *et al.* 2006), sendo generalizado, mais tarde, por Pareto (1896).

Definição 2 (Dominância de Pareto): Para um problema de minimização, um vetor $\mathbf{x} = (x_1, ...x_k)$ domina um outro vetor $\mathbf{y} = (y_1, ..., y_k)$ se e somente se $\mathbf{x}$ é menor ou igual a $\mathbf{y}$ em todas as dimensões, mas é menor em pelo menos uma, por exemplo, $\forall_i \in \{1, ..., k\}$, $x_i \leq y_i$ e $\exists i \in \{1, ..., k\}$: $x_i < y_i$. O símbolo $\prec$ expressa a relação de dominância, se temos que $\mathbf{x} \prec \mathbf{y}$, dizemos que $\mathbf{x}$ domina $\mathbf{y}$.

Definição 3 (Otimalidade de Pareto): Uma solução $\mathbf{x} \in \Omega$, onde $\mathbf{u} = F(\mathbf{x}) = f_1(\mathbf{x}), ..., f_k(\mathbf{x})$, é dita ótimo de Pareto se e somente se não existe uma solução $\mathbf{x'} \in \Omega$, onde $\mathbf{v} = F(\mathbf{x'}) = f_1(\mathbf{x'}), ..., f_k(\mathbf{x'})$, tal que $\mathbf{v} \prec \mathbf{u}$.

O conceito de otimalidade de Pareto é de extrema importância na resolução de MOPs, a seguir são apresentados outros conceitos que se fazem necessários quando se trabalha com otimização multiobjetivo.

Definição 4 (Conjunto Ótimo de Pareto): Para um dado problema multiobjetivo, o conjunto ótimo de Pareto, P^*, é definido por:

$$P^* = \{\mathbf{x} \in \Omega | \neg \exists \mathbf{x'} \in \Omega, F(\mathbf{x'}) \prec F(\mathbf{x})\}.$$

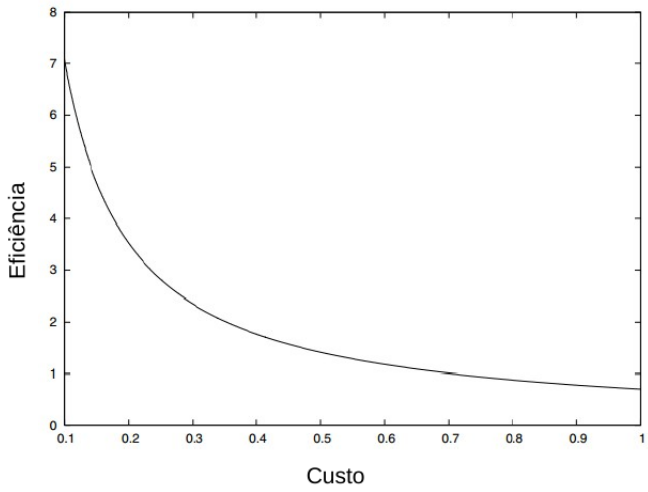
Definição 5 (Fronteira de Pareto): Para um dado problema multiobjetivo e um conjunto ótimo de Pareto, P^*, a fronteira de Pareto é definida por:

$$PF^* = \{\mathbf{u} = F(\mathbf{x}) | \mathbf{x} \in P^*\}.$$

O tomador de decisão seleciona as soluções com base no desempenho obtido para cada objetivo, o que é representado pela Fronteira de Pareto. Escolher uma solução que otimiza apenas um objetivo pode levar ao caso em que soluções que são melhores, de um ponto de vista geral, não sejam obtidas. O conjunto ótimo de Pareto contém aquelas soluções que são melhores. O objetivo de um algoritmo de otimização baseado em meta-heurísticas é encontrar um conjunto de soluções que se aproxima do conjunto ótimo de Pareto. O conjunto de soluções encontrado por um algoritmo heurístico é chamado de conjunto aproximado e sua imagem do espaço de objetivos é chamada de fronteira de Pareto aproximada.

Um exemplo de uma fronteira de Pareto é mostrado na Figura 4. Nessa situação, o melhor valor possível seria o máximo de eficiência e um valor mínimo de custo. Porém, como esses objetivos são conflitantes, esse ponto é utópico, ou seja, numa situação real o engenheiro não vai conseguir a maior eficiência com custo zero. Percebe-se que não é possível obter o melhor valor possível para os dois objetivos, pois geralmente eles são conflitantes. Assim, quando um objetivo melhora o seu valor, o outro piora.

Figura 4 – Exemplo de problema com duas funções objetivo: custo e eficiência.

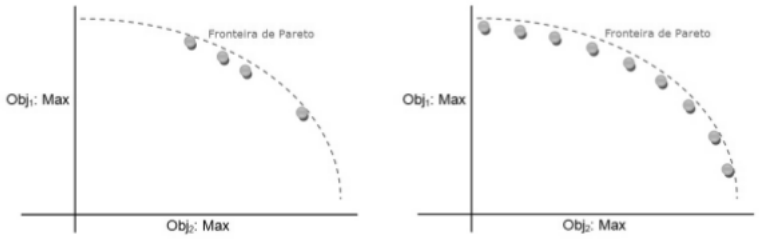


Fonte: Adaptado de Coello, Lamont e Veldhuizen (2006)

Além das definições apresentadas anteriormente, é interessante que se tenha em mente os conceitos de convergência e diversidade. Em linhas gerais, Coello, Lamont e Veldhuizen (2006) definem a convergência da fronteira de Pareto aproximada para a fronteira de Pareto real como a aproximação do conjunto de soluções obtido até a iteração atual do algoritmo que resolve um determinado MOP, até a fronteira de Pareto real. A aproximação das fronteiras de Pareto real e aproximada é algo interessante no que diz respeito à qualidade das soluções encontradas, mas não é o único ponto a ser observado. É importante que as soluções obtidas estejam espalhadas

em diversas regiões do espaço de busca; é justamente nesse ponto que se observa a diversidade em relação à fronteira de Pareto. As figuras 5(a) e 5(b) mostram um exemplo de um conjunto de aproximação com uma boa convergência, mas com uma diversidade ruim e um exemplo de um conjunto de aproximação com uma boa diversidade, respectivamente.

Figura 5 – Representação de conjuntos com boa convergência e boa diversidade. Os pontos indicam as fronteiras aproximadas.



(a) Boa convergência (b) Boa diversidade

Fonte: (CARVALHO, 2013)

3.2 Algoritmos evolucionários

Uma forma de lidar com problemas multiobjetivo é usar Algoritmos Evolucionários Multiobjetivo (MOEA, do inglês, *Multi-Objective Evolutionary Algorithms*) (COELLO; LAMONT; VELDHUIZEN, 2006). Visando garantir que as soluções escolhidas sejam, realmente, as melhores encontradas durante o processo de busca, os MOEAs fazem uso dos conceitos da otimalidade de Pareto. Para que um MOEA cumpra, de forma eficiente, a tarefa que lhe foi designada, é necessário que ele seja dotado que mecanismos que favoreçam a diversidade das soluções, evitando assim, que o algoritmo venha a convergir para apenas uma solução.

Dentre os algoritmos evolucionários encontra-se o *Elitist Non-dominated Sorting Genetic Algorithm* (NSGA-II). Apresentado em (DEB et al., 2002b), o NSGA-II é um algoritmo genético que usa elitismo e busca manter a diversidade populacional durante o processo de busca por soluções. Na busca, o NSGA-II considera uma população de cromossomos (as soluções) formados por gens (elementos que compõem as soluções). Os cromossomos são modificados através de operadores genéticos para que uma nova população, melhor que a população anterior, seja criada. Os operadores genéticos variam de acordo com o tipo de problema ao qual o algoritmo está sendo aplicado.

Além do NSGA-II, outro algoritmo evolucionário largamente estudado na literatura é a Otimização por Enxame de Partículas (PSO, do inglês *Particle Swarm Optimization*).O PSO é uma meta-heurística populacional baseada no comportamento de grupos de pássaros. A meta-heurística explora um espaço de busca n-dimensional utilizando um conjunto de soluções que é atualizado a cada geração. Cada solução é uma partícula e o conjunto de soluções é

chamado de nuvem. Essa nuvem se movimenta pelo espaço de busca de forma cooperativa. Cada partícula se move seguindo regras simples, considerando um operador de velocidade (KENNEDY; EBERHART, 2001). Dentre os algoritmos baseados em PSO, destaca-se o SMPSO, apresentado em (NEBRO et al., 2009). O algoritmo proposto pelo autor segue os passos de um algoritmo PSO multiobjetivo, a sua principal característica é limitar a velocidade das partículas, isso é feito para que as partículas sejam impedidas de percorrer regiões do espaço de busca que contenham soluções ruins.

Outros algoritmos que vêm sendo utilizados na resolução de problemas de otimização são os *Estimation of Distribution Algorithms*. Os EDA são baseados nos algoritmos genéticos, entretanto, não utilizam operadores de mutação, ao invés disso, um modelo probabilístico é induzido e utilizado na construção de novas populações. Os EDA estão começando a ser explorados em problemas multiobjetivo e serão o foco desse trabalho. Mais detalhes sobre os EDA são apresentados a seguir.

3.2.1 *Estimation of distribution algorithms*

De acordo com Bengoetxea (2002), de maneira geral, todas as estratégias de busca podem ser classificadas em dois tipos: completas ou heurísticas. A diferença entre esses dois tipos de estratégias consiste no fato de uma busca completa examinar sistematicamente todas as soluções pertencentes ao espaço de busca, enquanto as estratégias heurísticas se concentram apenas em uma parte dessas soluções.

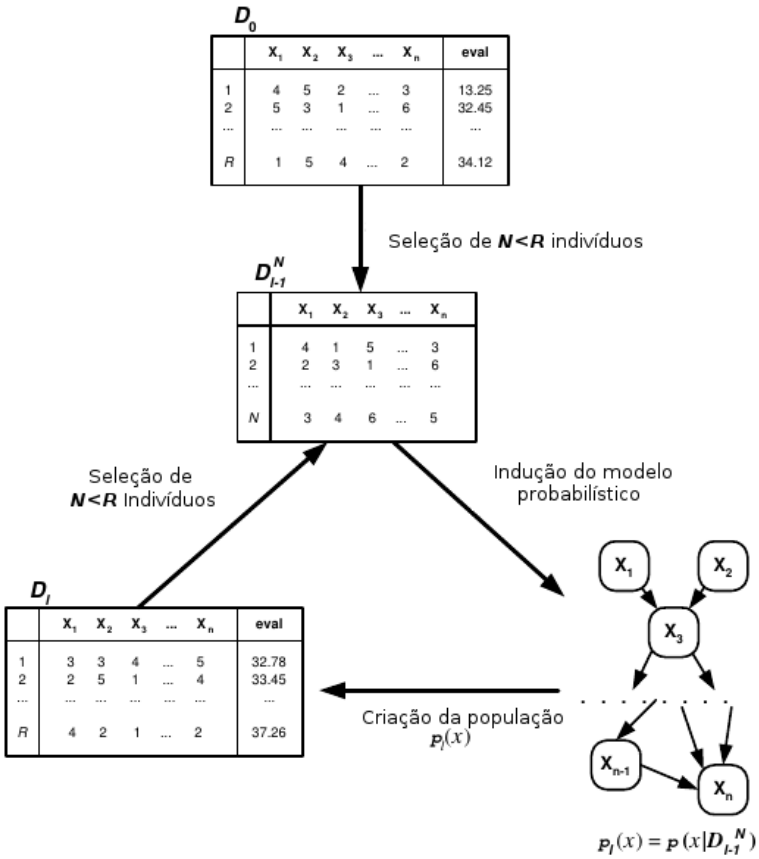
As estratégias heurísticas são classificadas em dois grupos: determinísticas e não-determinísticas. A principal característica das estratégias determinísticas é que, sob as mesmas condições, sempre será obtida a mesma solução; estratégias não-determinísticas fazem uso de artifícios que garantem ao algoritmo alguma aleatoriedade, fazendo com que, diferentes execuções possam levar a diferentes resultados sob as mesmas condições.

Algumas heurísticas armazenam apenas uma solução por iteração, outras armazenam um conjunto de soluções. No segundo caso, cada solução é chamada de indivíduo e o conjunto de soluções é chamado de população. Nestas heurísticas, a população evolui a cada iteração, de forma a atingir áreas mais promissoras do espaço de busca. Os algoritmos genéticos são exemplos desse tipo de heurística.

O comportamento dos algoritmos genéticos depende da definição de um grande número de parâmetros e esses algoritmos apresentam um desempenho ruim em problemas onde o critério utilizado na evolução das populações é perdido com o passar do tempo. Diante disso, surgiu a motivação de criar um novo tipo de algoritmo, chamado de *Estimation of Distribution Algorithm*. Nos EDA, a nova população é gerada sem a necessidade de utilizar operadores de *crossover* e mutação. Os novos indivíduos são gerados utilizando uma distribuição de probabilidade obtida através da seleção de indivíduos da população anterior (BENGOETXEA, 2002). Na Figura 6, é

apresentado um esquema onde é representado o funcionamento de um EDA.

Figura 6 – Representação gráfica do funcionamento de um EDA.



Fonte: Adaptado de (BENGOETXEA, 2002).

De maneira geral, na execução de um EDA são realizadas as seguintes etapas:

1. Uma população inicial D_0 com R indivíduos é gerada aleatoriamente.
2. Para gerar a população D_l , N indivíduos da população D_{l-1} são selecionados seguindo algum critério de seleção.
3. Um modelo probabilístico que melhor representa as interdependências entre as variáveis do problema é induzido a partir dos N indivíduos selecionados no passo anterior.
4. A nova população D_l , constituída por R indivíduos gerados através da simulação da distribuição de probabilidade aprendida no passo 2.

Os passos 2, 3 e 4 são repetidos até que algum critério de parada seja satisfeito.

Na literatura, são encontrados trabalhos onde os autores utilizam os EDA tanto para resolver problemas mono-objetivo quanto para resolver problemas multiobjetivo. Alguns destes trabalhos são apresentados a seguir.

Em (PAUL; IBA, 2002), são citadas diferentes variações dos EDA, utilizadas na resolução de problemas cujas soluções são binárias e permutações, e classificadas de acordo com

o número de dependências existentes entre as variáveis do problema. Um problema onde as variáveis são independentes é aquele onde não existe qualquer dependência entre as variáveis de decisão. Um algoritmo utilizado para resolver esse tipo de problema é o *Univariate Marginal Distribution Algorithm* (UMDA) (MÜHLENBEIN, 1997). Os EDA também são utilizados para resolver problemas onde as dependências entre as variáveis são bivariadas, isto é, existem, no máximo, duas dependências entre as variáveis. Um exemplo de EDA utilizado na resolução desse tipo de problema é o *Bivariate Marginal Distribution Algorithm* (BMDA) (PELIKAN; MUEHLENBEIN, 1999). Para resolver problemas onde as dependências entre as variáveis são multivariadas são utilizados, dentre outros algoritmos, o *Factorized Distribution Algorithm* (FDA)(MUHLENBEIN; MAHNIG, 1999) e o *Bayesian Optimization Algorithm* (BOA) (PELIKAN; GOLDBERG; CANTÚ-PAZ, 2000).

Em (TSUTSUI, 2002) é apresentado um EDA utilizado para resolver o Problema do Caixeiro Viajante (TSP, do inglês *Traveling Salesman Problem*). O algoritmo inicia gerando aleatoriamente uma permutação para cada indivíduo da população, a partir disso, soluções promissoras são selecionadas utilizando um método de seleção qualquer. Uma matriz de histograma de arestas (EHM, do inglês *Edge Histogram Matrix*) é construída a partir das soluções selecionadas e as novas soluções são geradas a partir da EHM.

Em (PELIKAN; SASTRY; GOLDBERG, 2006) são apresentados conceitos utilizados em EDA multiobjetivo, além disso, é apresentado o algoritmo mBOA, que utiliza redes *bayesianas* na construção do modelo probabilístico.

3.3 Otimização com muitos objetivos

Quando se lida com problemas complexos é comum se deparar com situações onde mais de quatro objetivos objetivos são considerados. Um exemplo disso é o problema investigado neste trabalho, onde podemos mapear até onze funções objetivo caso sejam consideradas as métricas apresentadas na Tabela 2. Fazer com que os algoritmos existentes sejam escalados para conseguir tratar problemas que possuem muitos objetivos não é tarefa das mais simples. Deb e Jain (2014) elencam algumas das dificuldades encontradas nessa área quando são considerados algoritmos que trabalham segundo o princípio da relação de dominância entre as soluções:

- Grande parte das soluções da população é não-dominada: com o aumento do número de objetivos, o número de soluções não-dominadas também aumenta. Uma vez que os MOEAs destacam as soluções não-dominadas na geração de uma nova população, um número alto de soluções não-dominadas faz com que se perca o critério nesse processo de geração, ocasionando uma desaceleração na busca;
- A avaliação de uma métrica de diversidade se torna computacionalmente custosa: analisar como as soluções estão dispostas no espaço de busca se torna difícil, isso porque é

computacionalmente custoso identificar uma vizinhança em um espaço de busca que possui muitas dimensões;

- Operações de recombinação podem ser ineficientes: se poucas soluções são encontradas em um espaço de busca amplo, a probabilidade de que estas soluções estejam em pontos distantes umas das outras é alta, assim como é alta a probabilidade de que a aplicação de operações de recombinação em soluções situadas em pontos distantes gere soluções também distantes das soluções-pai. Dessa maneira, é necessário que operadores especiais de recombinação sejam utilizados quando se trabalha com problemas que possuem muitos objetivos;

- A visualização é difícil: embora não seja um ponto relacionado à otimização propriamente dita, visualizar um espaço composto por um alto número de dimensões pode ser uma tarefa complexa.

Diante dessas dificuldades, viu-se a necessidade de estudar maneiras de construir algoritmos evolucionários para serem aplicados em problemas com um número alto de objetivos, surgindo assim a Otimização com Muitos Objetivos. Enquanto a otimização multiobjetivo lida com problemas que possuam até três objetivos conflitantes, a otimização com muitos objetivos é utilizada na resolução de problemas que levam em consideração quatro ou mais objetivos (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Nessa área são propostos novos métodos que visam reduzir a deterioração da busca à medida que o número de objetivos do problema aumenta.

Para que o impacto dos problemas anteriormente apresentados seja reduzido, algumas abordagens têm sido apresentadas na literatura (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008), dentre elas estão:

- Adaptação das relações de preferência: nessa abordagem, a pressão em direção à fronteira de Pareto é maximizada, para fazer isso, uma nova ordenação do espaço de objetivos é realizada através da definição de novas relações de preferências.

- Redução da dimensionalidade: a ideia consiste em identificar os objetivos menos conflitantes entre si para que possam ser removidos, de modo que o conjunto ótimo de Pareto não seja alterado.

- Estratégias de decomposição: utiliza-se métodos de decomposição para dividir o problema multiobjetivo em problemas escalares, após isso, algoritmos evolucionários são usados para resolver os problemas menores.

- Incorporação de preferências: a ideia desse método considera informações de preferência que são utilizadas para direcionar a busca para regiões específicas da fronteira de Pareto.

- Diferentes esquemas de avaliação das soluções: nessa abordagem, a dominância de Pareto não é utilizada na avaliação das soluções, ao invés disso, são utilizadas outras medidas de avaliação.

Dentre as várias abordagens exploradas nessa área, vale destacar a utilização de métodos de arquivamento. Métodos de arquivamento são utilizados para escolher quais das melhores soluções encontradas até o momento em um processo de busca ficarão armazenadas em um arquivo externo. Em (BRITTO; POZO, 2012), a ideia de um arquivador simples é definida como: dada uma sequência de pontos (vetores-objetivo gerados por um MOEA), o objetivo do arquivador – de tamanho N – é manter um subconjunto desses pontos em um arquivo A, tal que $|A| < N$. Os autores aplicam diferentes métodos de arquivamento ao MOPSO em problemas com muitos objetivos. Após a realização dos experimentos, concluem que o uso de arquivadores onde o tamanho do arquivo é ilimitado traz melhores resultados. No entanto, ponderam que nem sempre é possível recorrer a arquivos com tamanhos ilimitados. Verificam também que o arquivador ideal, proposto no trabalho, atinge bons resultados considerando a convergência em relação a fronteira de Pareto, entretanto, notam que as soluções encontradas se concentram na região próxima ao joelho da fronteira de Pareto, ou seja, o ponto de maior arqueamento da curva.

Além dos métodos de arquivamento, outra abordagem explorada para minimizar as dificuldades encontradas ao lidar com problemas de otimização com muitos objetivos é a utilização de um conjunto de pontos de referência. A ideia consiste em não realizar uma busca considerando a Fronteira de Pareto por completo, e sim realizar várias buscas menores em diferentes regiões da fronteira. Em (DEB; JAIN, 2014), é proposto um algoritmo que se baseia nesse princípio. O autores sugerem duas formas a serem utilizadas para definir como as buscas menores são realizadas: ou as buscas são realizadas levando em consideração diferentes direções da fronteira, ou é definido um conjunto de pontos de referência estruturados em um hiperplano e as buscas são realizadas nas regiões onde esses pontos estão localizados. O uso desta abordagem garante a preservação da diversidade das soluções.

O trabalho apresentado em (BRITTO; POZO, 2014) utiliza as duas abordagens descritas anteriormente. Os autores propõem um algoritmo (REF-I-MOPSO) baseado em PSO. A ideia principal do algoritmo é permitir que a busca seja realizada em diferentes áreas do espaço de objetivos previamente escolhidas pelo usuário. O algoritmo conta com um arquivador, o método de arquivamento utilizado é guiado por um ponto de referência previamente escolhido. Este ponto de referência faz parte de um hiperplano que representa uma região boa para a qual o algoritmo deve convergir. O algoritmo proposto por Britto e Pozo foi testado em vários problemas de otimização com muitos objetivos e verificou-se que as soluções geradas se encontravam próximas ao ponto de referência selecionado.

3.4 Considerações finais

Neste capítulo foram apresentados conceitos relacionados à otimização com muitos objetivos, classe na qual se enquadra o problema de busca por sequências de refatorações, que é investigado neste trabalho. Considerando o fato de que os EDA ainda não foram explorados no contexto de SBSR, o objetivo do trabalho é verificar se os EDA (detalhados no capítulo seguinte) são úteis na busca por sequências de refatorações. Tendo conhecimento das dificuldades enfrentadas ao se trabalhar com problemas que consideram muitos objetivos, foram utilizados métodos de arquivamento em conjunto com o EDA proposto, considerando que a utilização desses métodos se mostrou eficaz em outros problemas com muitos objetivos.

4

O algoritmo EDA no contexto de busca por sequências de refatorações

Neste capítulo, são discutidas abordagens através das quais o EDA pode ser adaptado para resolver problemas de otimização com muitos objetivos de tal maneira que seja possível a sua aplicação na busca por sequências de refatorações, problema apresentado no Capítulo 2.

4.1 Visão geral do algoritmo

Neste trabalho foi adotada a representação do problema apresentada na Seção 2.2.1, na qual uma sequência de refatorações R_i é aplicada a um software S, objetivando alterar a condição do software de um determinado estado S_i para um estado S_j.

A fim de mensurar e comparar as melhorias obtidas nas características internas do software decorrentes da aplicação das sequências de refatorações foram definidas funções objetivo. Dada a natureza do problema, as funções objetivo consideradas na avaliação das soluções são mapeadas como métricas da Engenharia de Software. Neste trabalho, os seis atributos de qualidade do modelo QMOOD (detalhado na Seção 2.2.2) são adotados como funções objetivo; a escolha desse conjunto de métricas foi motivada pela relevância deste modelo, evidenciada pela sua larga utilização em outros trabalhos encontrados na literatura ((O'KEEFFE; CINNÉIDE, 2006),(O'KEEFFE; CINNÉIDE, 2008a),(O'KEEFFE; CINNÉIDE, 2008b)).

Após o mapeamento do problema de refatoração como um problema de otimização e da definição das funções objetivo, foi preciso escolher uma ferramenta que provesse a infraestrutura necessária para a realização dos experimentos. Por ser uma ferramenta *open source* onde é possível a realização de adaptações (como inclusão de algoritmos e problemas), o *framework* descrito na Seção 2.2.4 foi escolhido. Neste *framework* estão implementadas as seguintes refatorações: *Pull Up Method, Pull Up Field, Push Down Method, Push Down Field*, *Self Encapsulate Field* e *Increase Method*. Detalhes sobre estas refatorações são apresentados no Apêndice A.

Para utilizar o EDA na resolução deste problema, cada sequência de refatorações foi mapeada como uma sequência de números inteiros, onde cada número representa uma refatoração do conjunto de refatorações possíveis, encontradas pelo módulo de refatoração do *framework*. Ao relacionar cada refatoração a um número inteiro, a solução do problema passa a ser uma permutação de valores inteiros. Para que fosse possível a inclusão do EDA proposto no *framework*, o jMetal (DURILLO; NEBRO, 2011) foi utilizado na implementação do algoritmo.

Durante o processo de busca, dois elementos impactam diretamente no desempenho do EDA: o modelo probabilístico e o método de seleção utilizados. O modelo probabilístico é responsável por representar características existentes em uma população, de modo que estas características possam ser utilizadas na geração das próximas populações. Por exemplo: quais valores estão sendo atribuídos a determinadas variáveis de decisão e como isto reflete na qualidade das soluções. O método de seleção é responsável por identificar quais soluções da população atual serão utilizadas na geração do modelo probabilístico. Os métodos de seleção utilizados variam de acordo com as características dos problemas que estão sendo tratados, neste trabalho, métodos de arquivamento foram utilizados na seleção das soluções. Existem diversos métodos de arquivamento e o que os diferencia é o critério de decisão utilizado para determinar se uma solução é melhor que outra. Mais detalhes sobre os modelos probabilísticos e métodos de arquivamento utilizados neste trabalho são apresentados nas seções a seguir.

Utilizando o modelo probabilístico e o método de seleção o EDA consegue identificar quais novas sequências de refatorações farão parte da nova população. Estas refatorações são aplicadas e, tendo como base as métricas da Engenharia de Software, verifica-se o quanto a aplicação dessas refatorações melhorou as características do software. Este ciclo de criação de populações e avaliação das soluções é repetido até que um critério de parada seja satisfeito.

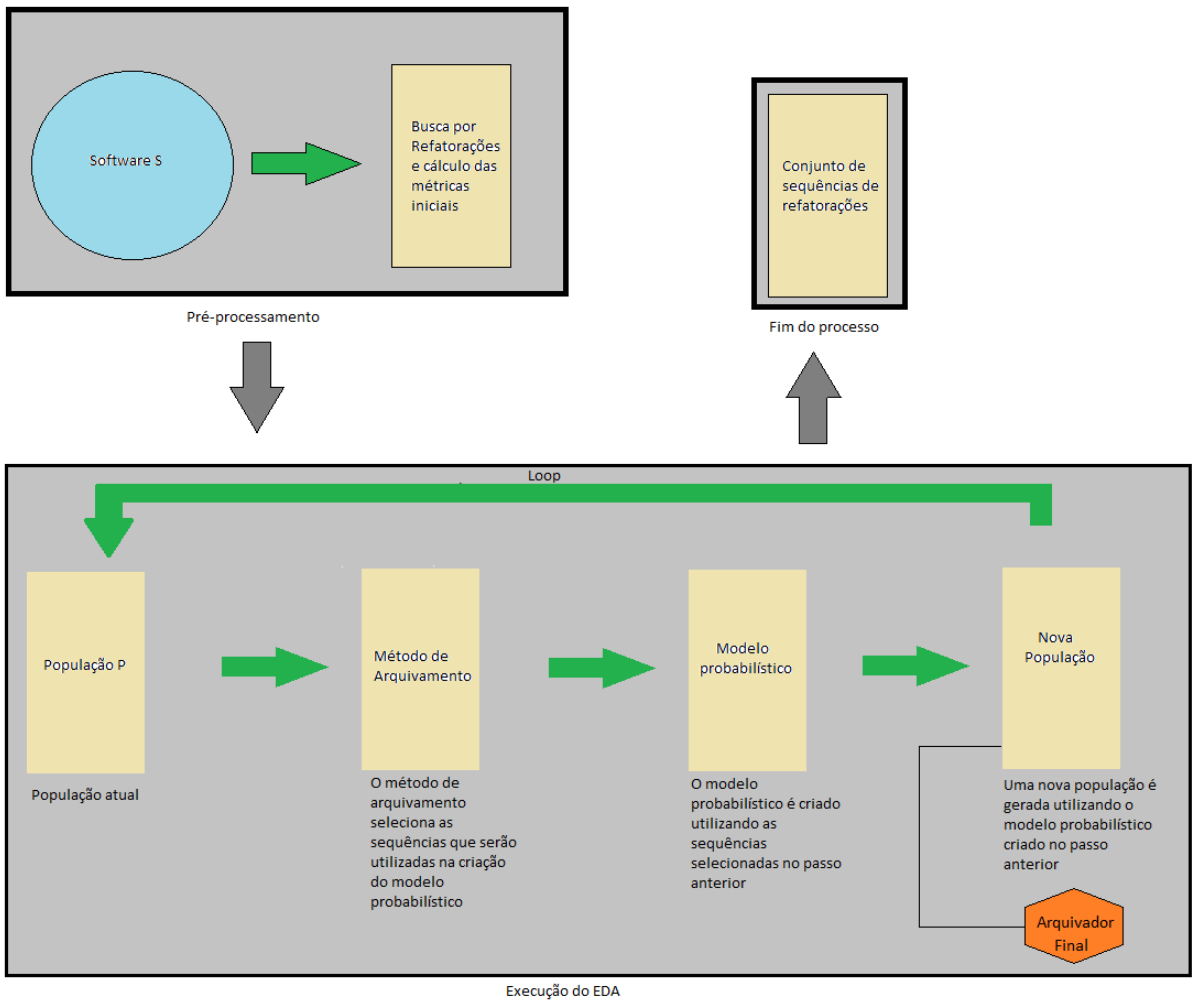
4.1.1 Visão geral do algoritmo proposto

Uma vez apresentados os elementos necessários para a execução do EDA, bem como o mapeamento do problema de refatoração como um problema de otimização e a definição das funções objetivo, pode-se ter uma ideia do que ocorre em uma iteração da execução do EDA. A iteração segue o fluxo apresentado na Figura 7.

O início da iteração se dá com a leitura de um software S pelo *framework* de refatoração. O código-fonte do software é lido e o módulo de refatoração do *framework* realiza a busca por refatorações válidas e calcula as métricas iniciais do software. Após as refatorações válidas e as métricas iniciais serem calculadas, a execução do EDA é iniciada no módulo de busca do *framework*.

Na Figura 7 é mostrado que o método de arquivamento seleciona as soluções – no contexto de SBSR, cada solução é computacionalmente representada por uma sequência de números inteiros, onde cada número representa uma refatoração – que serão utilizadas na geração

Figura 7 – Fluxo de execução do EDA.



Fonte: Produzido pelo autor.

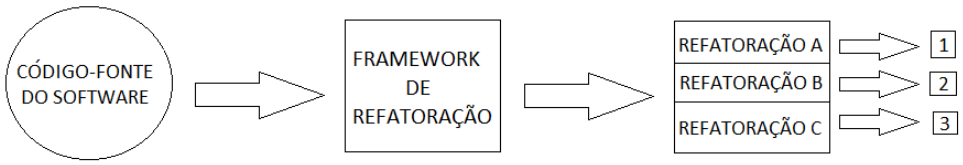
do modelo probabilístico. Pode acontecer de o método de arquivamento precisar escolher entre retirar uma solução do arquivador para adicionar uma nova solução, caso o arquivador esteja cheio. O critério de decisão para isto varia de acordo com o método de arquivamento. Após a seleção das soluções, o modelo probabilístico é gerado, e posteriormente utilizado na criação de uma nova população. Este ciclo é repetido até que um critério de parada seja satisfeito. Ao fim da execução, é retornado um conjunto de sequências de refatorações de software.

Codificação da solução

Para que o EDA pudesse ser aplicado no contexto de SBSR, as refatorações foram mapeadas como números inteiros. No início do processo, o código-fonte do projeto a ser refatorado é lido pelo *framework* de refatoração automática de software. Após a leitura do código-fonte, o módulo de refatoração do *framework* identifica quais são as refatorações válidas, que poderão ser consideradas na fase de busca por sequências de refatorações. Uma vez identificadas

as refatorações, cada uma delas é identificada por um número inteiro pelo EDA. Dessa maneira, para o algoritmo, uma sequência de refatorações é uma permutação de números inteiros. O fluxo deste processo é apresentado na Figura 8.

Figura 8 – Mapeamento das refatorações.



Fonte: Produzido pelo autor.

Criação da população inicial

No início da execução, o algoritmo não possui informações para estimar um modelo probabilístico e, a partir dele, criar uma população. Diante disso, a população inicial é formada por sequências cujas refatorações são escolhidas aleatoriamente.

Conjunto de soluções final

Conforme dito anteriormente, ao final da execução do algoritmo é retornado um conjunto de sequências de refatorações. Esse conjunto de sequências é mantido em um arquivo atualizado ao final de cada iteração do algoritmo, ou seja, sempre que uma nova população é gerada, verifica-se a possibilidade de adicionar as soluções que a compõe neste arquivo. Assim, ao final da execução, as melhores soluções não-dominadas – de acordo com o método de arquivamento utilizado – ficam armazenadas no arquivo. Este arquivo não é o mesmo utilizado na etapa de seleção das soluções que serão utilizadas na geração do modelo probabilístico, e possui uma capacidade maior. Nos experimentos realizados neste trabalho, o arquivo que guarda as soluções finais sempre tem capacidade para guardar 100% do tamanho da população considerada pelo algoritmo.

4.2 Modelos probabilísticos utilizados

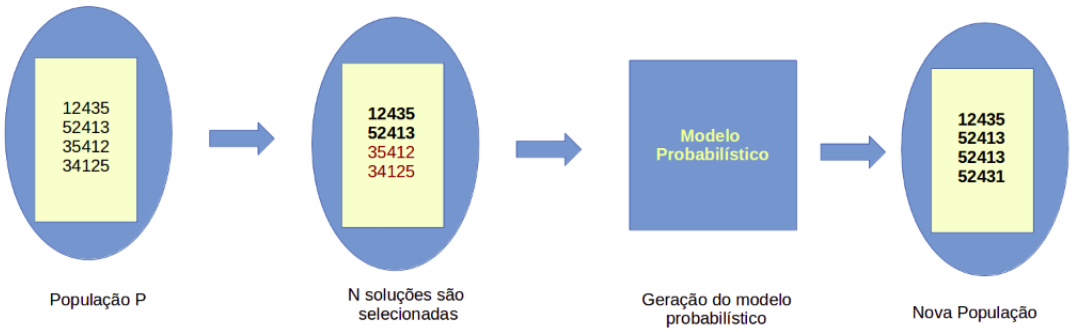
Conforme apresentado na Seção 3.2.1, a execução de um EDA segue um fluxo bem definido: criação da população inicial, seleção de soluções para criação da próxima população, geração da nova população com base em um modelo probabilístico específico. O processo se repete até que um critério de parada seja satisfeito.

O modelo probabilístico utilizado varia de acordo com o problema a ser resolvido. Durante o desenvolvimento deste trabalho, foram utilizados dois modelos probabilísticos que objetivam a resolução de problemas onde as soluções são permutações. Estes modelos foram escolhidos tendo em vista os bons resultados obtidos por eles em problemas cuja estrutura da solução se assemelha à estrutura da solução do problema tratado neste trabalho.

4.2.1 *Univariate Marginal Distribution Algorithm*

O primeiro modelo probabilístico utilizado é apresentado em (MÜHLENBEIN, 1997). Chamado de *Univariate Marginal Distribution Algorithm* (UMDA), o algoritmo em questão foi projetado para resolver problemas onde não exista dependência entre as variáveis de decisão. Para gerar uma nova população o modelo probabilístico conta quantas vezes determinada variável de decisão recebeu um valor v considerando as soluções da população atual. A nova população é gerada a partir destas frequências. As frequências são utilizadas da seguinte forma: considerando uma população P, formada por N soluções do tipo $x_0, x_1, ..., x_n$, é calculado o número de vezes em que uma variável de decisão recebeu o valor v, este cálculo é feito para todas as variáveis de decisão e para todos os valores que podem ser atribuídos a uma variável de decisão. Uma vez calculadas as frequências, os valores obtidos servem de base para a geração da próxima população. Por exemplo, se a variável x_0 recebeu o valor v em 40% das soluções da população atual, a probabilidade da variável x_0 receber o valor v em uma solução da nova população é de 40%. Na Figura 9 é apresentada uma esquematização de como o modelo funciona.

Figura 9 – Representação gráfica do funcionamento do UMDA.



Fonte: Produzido pelo autor.

Na Figura 9, cada variável de decisão que compõe uma solução ocupa uma posição no vetor e, para que seja gerada uma nova população, são calculadas as frequências em que cada variável recebe os valores 1, 2, 3, 4 e 5. No exemplo, as duas primeiras soluções são selecionadas, considerando estas soluções, percebe-se que a primeira variável recebeu o valor 1 uma vez e recebeu o valor 5 uma vez, ou seja, 50% das soluções selecionadas tiveram a primeira variável definida como 1 e os outros 50% das soluções tiveram a primeira variável definida como 5; a partir disso, no processo de geração da nova população, a probabilidade de uma solução ter a primeira variável definida como 1 ou 5 é de 50% para cada um destes valores. Este processo é repetido para todas as variáveis de decisão.

Em (PAUL; IBA, 2002), é apresentada uma versão do UMDA onde, além do cálculo das frequências, o autor utiliza uma modificação probabilística. Na modificação probabilística, quando algum valor é selecionado para uma variável, as probabilidades desse valor ser utilizado

nas próximas variáveis caem para 0 e as probabilidades dos valores ainda não selecionados aumentam proporcionalmente. No exemplo apresentado na Tabela 5 e na Tabela 6 é definido um valor para a variável X_1.

Tabela 5 – Antes da seleção de X_1.

Variável/Valor	1	2	3
X_1	0,7	0,1	0,5
X_2	0,1	0,6	0,1
X_3	0,2	0,3	0,4

Fonte: Adaptado de (PAUL; IBA, 2002)

Tabela 6 – Após a seleção de X_1.

Variável/Valor	1	2	3
X_1	0,7	0	0
X_2	0,1	0,65	0,35
X_3	0,2	0,35	0,65

Fonte: Adaptado de (PAUL; IBA, 2002)

Se X_1 recebe o valor 1, então as probabilidades de X_1 receber o valor 2 ou o valor 3 são nulas e as probabilidades das variáveis X_2 e X_3 receberem estes valores aumentam proporcionalmente. No exemplo mostrado, antes de receber o valor 1, a probabilidade de X_1 receber o valor 2 era $0,1$, como X_1 já recebeu um valor, a chance de atribuir o valor 2 a esta variável cai para zero e o valor da probabilidade anterior é dividido proporcionalmente e adicionado às probabilidades das outras variáveis receberem o valor 2. Desta maneira, as probabilidades de X_2 e X_3 receberem o valor 2 aumentam em $0,05$. O mesmo cálculo é feito para o valor 3, onde a probabilidade inicial de X_1 é 0,5. Após a definição do valor de X_1, as probabilidades de X_2 e X_3 receberem o valor 3 aumentam em 0,25.

4.2.2 *Edge Histogram Matrix*

Além do UMDA, foi utilizado o modelo probabilístico baseado no que foi apresentado em (TSUTSUI, 2002), onde o autor propõe um EDA para resolver o TSP. O primeiro passo do algoritmo é gerar uma população inicial, no geral, a sequência de variáveis de decisão é uma permutação de valores inteiros, gerada aleatoriamente. Após a criação da população, algumas soluções são escolhidas utilizando um método de seleção qualquer. Uma vez que as soluções são selecionadas, o próximo passo é montar uma Matriz de Histograma de Arestas (EHM, do inglês *Edge Histogram Matrix*) utilizando estas soluções.

Para construir a EHM, considera-se que $s_k^t = (\pi_k^t(0), \pi_k^t(1), ..., \pi_k^t(L-1))$ seja o k-ésimo indivíduo da população $P(t)$ na iteração t. Onde $s_k^t = (\pi_k^t(0), \pi_k^t(1), ..., \pi_k^t(n))$ são permutações de $(0, 1, ..., L-1)$ e L é o tamanho da permutação. A EHM^t da população $P(t)$ consiste em L^2 elementos, $(e_{ij}^t)(i, j = (0, 1, 2, ..., L-1))$, obtidos da seguinte maneira:

$$e^t_{i,j} = \begin{cases} \sum_{k=1}^{N}(\delta_{i,j}(s^t_k) + \delta_{j,i}(s^t_k)) + \varepsilon, & se\ i \neq j \\ 0, & \text{caso contrário} \end{cases} \quad (4.1)$$

onde N é o tamanho da população e $\delta_{i,j}$ é uma função definida por

$$\delta_{i,j} = \begin{cases} 1, & se\ \exists h[h \in 0,1,...,L-1 \wedge \pi^t_k(h) = i \wedge \pi^t_k((h+1) mod L) = j] \\ 0, & \text{caso contrário} \end{cases} \quad (4.2)$$

Considerando que i e j sejam valores que podem ser atribuídos a uma variável de decisão, de acordo com a Equação 4.2, o valor de $\delta_{i,j}$ é definido como 1 apenas se os valores i e j são encontrados em posições consecutivas de uma permutação. O valor do $\delta_{i,j}$ influencia diretamente na definição dos elementos da matriz, quanto mais indivíduos tiverem os valores i e j em posições consecutivas, maior vai ser o valor de $e^t_{i,j}$, definido na Equação 4.1, em outras palavras, maior a chance de variáveis que ocupam posições consecutivas no vetor receberem os valores i e j no momento de criação da nova população. A ideia por trás disso é que, com o passar das iterações o algoritmo aprenda a ordem de valores que devem ser atribuídos às variáveis de decisão para que sejam geradas soluções melhores.

O *bias* $\varepsilon = \frac{2N}{L-1} B_{ratio}$ é utilizado para controlar a pressão no momento de gerar novas soluções. A atribuição de um menor valor para $B_{ratio}(B_{ratio} > 0)$ reflete a real distribuição das arestas no momento da criação de novas soluções.

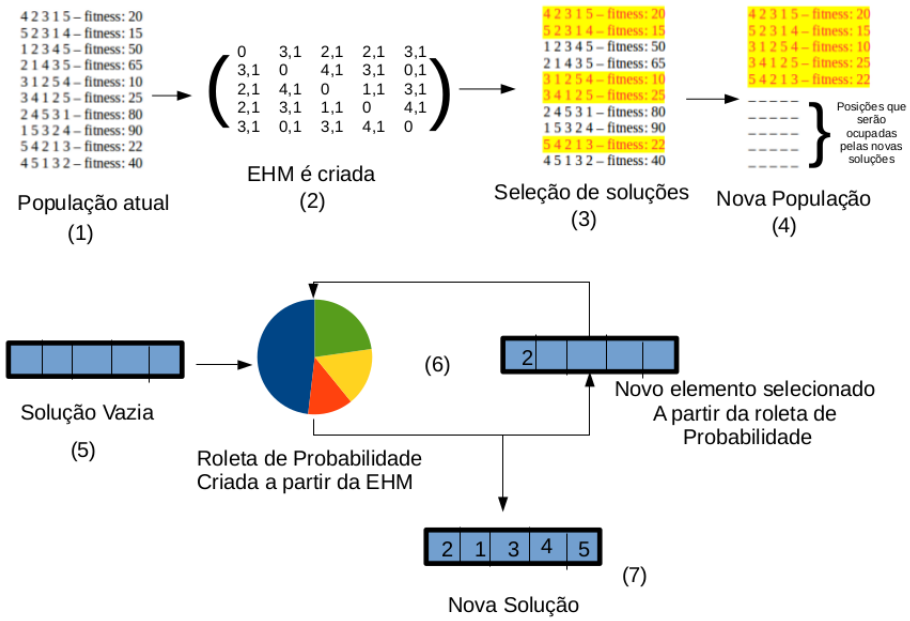
Para gerar uma nova solução c, o algoritmo segue os seguintes passos:

1. O contador de posições p é definido como zero.
2. O primeiro elemento da solução,$c[0]$, é selecionado aleatoriamente.
3. É construído um vetor-roleta rw a partir da EHM^t. Cada elemento do vetor é definido da seguinte forma: $rw[j] = e^t_{c[p],j}(j = (0,1,...,L-1))$.
4. Os elementos de rw que foram previamente escolhidos recebem o valor 0, de acordo com a seguinte regra: $rw[c[i]] = 0(i = 0,1,...,p)$.
5. O próximo elemento da solução, $c[p+1]$, é definido: $c[p+1] = x$, sendo que $x \in (y|rw[y] \neq 0)$ com a probabilidade $\frac{rw[x]}{\sum_{j=0}^{l-1} rw[j]}$.
6. Incrementar o valor de p em 1 unidade.
7. Se $p < L-1$, retornar ao passo 3.
8. Retornar a solução c.

O processo de criação de uma nova solução é representado graficamente na Figura 10

Em (1) é apresentada a população atual, nesta população, cada solução possui um valor de *fitness*, o *fitness* de uma solução está associado à qualidade dela, para um problema de minimização, quanto menor o *fitness*, melhor a solução; na etapa (2), a partir da Equação 4.1, é criada a EHM. Por exemplo, considerando que o algoritmo esteja na iteração 0, para calcular o

Figura 10 – Representação gráfica do funcionamento do EHM.



Fonte: Produzido pelo autor.

elemento $e^0_{3,5}$, é verificado, em todas as soluções da população, se os valores 3 e 5 são encontrados em posições consecutivas. No exemplo, percebe-se que este fato ocorre 3 vezes. Para cada uma dessas vezes, o valor de $e^0_{3,5}$ é acrescido de 1. Por fim, é adicionado a $e^0_{3,5}$ o valor do *bias* ε, que no exemplo mostrado é 0,1, resultando em $e^0_{3,5} = 3,1$. Em (3), algumas soluções da população atual são selecionadas utilizando um método de seleção aleatória. Após a seleção das soluções, uma nova população é criada e as soluções selecionadas são adicionadas a ela no passo (4). O próximo passo é a criação de uma solução vazia indicada em (5). Para preencher a solução, são seguidos os passos de criação de uma nova solução apresentados anteriormente nesta seção – a etapa (6) se repete até que todos os elementos da nova solução sejam definidos. Finalmente, na etapa (7), a nova solução está completa. Caso existam mais soluções a serem criadas na nova população, o processo se repete a partir da etapa (5).

4.3 Métodos de arquivamento utilizados

Conforme discutido no Capítulo 3, existem fatores que deterioram a qualidade do processo de busca dos MOEAs quando estes são utilizados para resolver problemas de otimização que possuem muitos objetivos. Dentre as várias estratégias utilizadas para minimizar essa deterioração está a utilização de métodos de arquivamento. Em linhas gerais, o objetivo de um método de arquivamento é manter em um arquivo de capacidade limitada as melhores soluções encontradas durante o processo de busca. Existem diversos métodos de arquivamento, o que os diferencia é o critério de decisão utilizado para determinar se uma solução é melhor que outra.

Para tornar o EDA proposto apto a resolver problemas que lidam com muitos objetivos, pensou-se em utilizar os modelos probabilísticos apresentados na seção anterior em conjunto com os métodos de arquivamento. Para isso, ao invés de utilizar um operador de seleção na escolha das soluções que serão utilizadas para estimar o modelo probabilístico que norteia a criação da próxima população, foram utilizados métodos de arquivamento, que trabalham no espaço dos objetivos para decidir quais as melhores soluções encontradas até então. Dessa maneira, as soluções mantidas no arquivo são aquelas utilizadas para estimar o modelo probabilístico.

Neste trabalho, foram investigados quatro métodos de arquivamento diferentes: o arquivador *Crowding Distance*, o arquivador Ideal, o arquivador SPEA2 e o *Multi-level Grid Archiving*. Conforme apresentado em (BOTELHO; BRITTO; SILVA, 2016), as definições destes arquivadores são as seguintes:

Arquivador *Crowding Distance*: Este arquivador utiliza a *Crowding Distance* (CD) para decidir quais soluções permanecerão no arquivo. A CD é utilizada para estimar a densidade de soluções ao redor de uma solução. Para calcular a CD é utilizada a distância média entre os pontos mais próximos em cada eixo de objetivo. A utilização deste arquivador introduz mais diversidade no processo de busca, uma vez que a função de seleção remove do arquivo as soluções com pior CD, isto é, as soluções localizadas nas regiões mais povoadas do espaço de objetivos.

Arquivador Ideal: O uso desta abordagem objetiva o aumento da convergência no processo de busca, fazendo com que as soluções no arquivo sejam direcionadas para uma área específica do espaço de objetivos. Para fazer isso, um ponto ideal é escolhido como guia. O ponto ideal é um vetor com o melhor valor para cada objetivo, este vetor é obtido considerando as soluções armazenadas no arquivo em cada iteração do algoritmo. A função de seleção calcula o ponto ideal e, a partir disso, calcula a distância euclidiana entre o ponto ideal e cada ponto pertencente ao arquivo. Após este cálculo, é removido do arquivo o ponto com a maior distância.

Arquivador SPEA2: A ideia por trás do arquivador SPEA2 é selecionar as soluções que estarão no arquivo de acordo com o cálculo de *fitness* realizado pelo algoritmo SPEA2 (ZITZLER; LAUMANNS; THIELE, 2002). A função de seleção utiliza a distância do *k-ésimo* vizinho mais próximo como medida de densidade. Nesta função, as soluções com piores distâncias são removidas do arquivo.

Multi-level Grid Archiving: Este arquivador combina o esquema do *grid* adaptativo com o arquivamento de Pareto. Se o arquivo estiver cheio, a função de seleção divide o espaço de objetivos em blocos e para cada ponto pertencente ao arquivo é definido um índice de bloco. Após isso, observa-se a relação de dominância entre os índices de blocos, caso a solução a ser adicionada pertença a algum dos blocos dominados, ela não é adicionada ao arquivo; caso contrário, um dos pontos que possuem blocos dominados é aleatoriamente escolhido e removido do arquivo. Caso não seja encontrada uma relação de dominância entre os blocos, o espaço de objetivos é novamente dividido em blocos menores até que uma relação de dominância seja

encontrada.

Os quatro métodos foram testados na fase de validação do algoritmo proposto. Após a análise dos resultados, decidiu-se utilizar apenas o arquivador *Crowding Distance* e o arquivador Ideal ao aplicar o EDA no problema de refatoração de software. Detalhes sobre os experimentos realizados são apresentados no Capítulo 5.

4.4 Considerações finais

Neste capítulo foram apresentados conceitos sobre os *Estimation of Distribution Algorithms* (EDA), além disso foram mostrados os modelos probabilísticos utilizados no desenvolvimento deste trabalho. Foi também discutida a adaptação realizada para que fosse possível utilizar o EDA na resolução de problemas com muitos objetivos. No contexto desse trabalho a abordagem utilizada foi a combinação do EDA com métodos de arquivamento, onde os métodos de arquivamento são utilizados na seleção das soluções que são utilizadas para compor o modelo probabilístico. Além disso, foi apresentada a forma como o EDA foi utilizado para resolver o problema de busca por sequências de refatorações de software.

5

Experimentos e Resultados

Para avaliar o desempenho do EDA desenvolvido, foi executado um conjunto de experimentos, divididos em duas fases: na primeira delas, o EDA foi validado em um problema de *benchmarking* para que sua adequabilidade à resolução de problemas com muitos objetivos pudesse ser verificada; na segunda fase, o algoritmo proposto foi utilizado para resolver o problema da busca por sequências de refatorações de software.

5.1 Metodologia de condução dos experimentos

Para que o algoritmo proposto pudesse ser avaliado foi definida uma metodologia a ser seguida durante a condução dos experimentos. Esta metodologia foi seguida nas duas fases dos experimentos:

- **Fase 1**: Validação do algoritmo;
- **Fase 2**: Aplicação do algoritmo no problema de busca por sequências de refatorações.

A primeira fase tem por objetivo verificar se o EDA obtém um bom desempenho em problemas com muitos objetivos, enquanto na segunda fase é avaliado o desempenho do algoritmo proposto no contexto de SBSR. Tanto na Fase 1 quanto na Fase 2 dos experimentos foram realizadas as seguintes atividades: configuração de parâmetros, execução dos experimentos e análise dos resultados. Detalhes sobre estas atividades são apresentados nas seções a seguir.

5.1.1 Configuração de parâmetros

Na etapa de configuração de parâmetros foi analisado o comportamento do algoritmo à medida que o número de iterações, tamanho da população e tamanho do arquivador utilizado variam.

Na Fase 1, foram consideradas populações com 100 e 200 indivíduos; para definir o tamanho do arquivador foram testados, para os dois tamanhos de população, seis tamanhos de arquivador: 5%, 10%, 20%, 30%, 40% e 50% do tamanho da população. Dessa forma, doze configurações de parâmetros foram testadas para cada um dos quatro métodos de arquivamento. Os métodos de arquivamento investigados na fase de validação foram: Arquivador *Crowding Distance*, Arquivador Ideal, Arquivador SPEA2 e *Multi-level Grid Archiving*. As variações de parâmetros testados na fase de validação do algoritmo são sumarizadas na Tabela 7. Os resultados obtidos nesta fase são apresentados na Seção 5.2.

Tabela 7 – Valores dos parâmetros testados na Fase 1.

Parâmetro	Valores
Tamanho da população	100 e 200
Tamanho do arquivador	5%, 10%, 20%, 30%, 40% e 50%
Métodos de arquivamento	*Crowding Distance*, Ideal, SPEA2 e *Multi-level Grid Archiving*

Fonte: Produzido pelo autor.

Considerando os resultados obtidos na fase de validação do algoritmo, o número de configurações de parâmetros foi reduzido na Fase 2. Nesta fase dos experimentos, o tamanho da população foi fixado em 100, após a percepção de que a qualidade das soluções que serão consideradas na geração do modelo probabilístico pode ser controlada apenas variando o tamanho do arquivador. A variação do tamanho do arquivador foi reduzida, uma vez que, na fase de validação do EDA, ficou perceptível uma queda de desempenho do algoritmo quando considerado um arquivador capaz de armazenar um número maior de soluções. Por este motivo, nesta fase dos experimentos, foram considerados três tamanhos de arquivador: 10%, 20%, 30% do tamanho da população. No que diz respeito aos métodos de arquivamento, percebeu-se, na fase de validação do algoritmo, que a variação dos métodos de arquivamento não trazia uma melhoria significativa aos resultados obtidos pelo EDA. Diante deste cenário, ao aplicar o EDA no problema de busca por refatorações, foram considerados apenas dois métodos de arquivamento, o arquivador *Crowding Distance* e o Arquivador Ideal, apresentados no Capítulo 4. As variações de parâmetros testados na fase de aplicação do algoritmo no contexto de SBSR são sumarizadas na Tabela 8.

Tabela 8 – Valores dos parâmetros testados na Fase 2.

Parâmetro	Valores
Tamanho da população	100
Tamanho do arquivador	10%, 20% e 30%
Métodos de arquivamento	*Crowding Distance* e Ideal

Fonte: Produzido pelo autor.

Além dos parâmetros citados anteriormente, foi necessário definir o número de avaliações de função objetivo, este número está relacionado à quantidade de soluções que serão avaliadas em uma execução completa do algoritmo. Este parâmetro foi fixado em 10.000 na Fase 1. Para

definir o valor deste parâmetro na Fase 2, foi realizada uma análise à parte, cujos detalhes são apresentados na Seção 5.3.1.

5.1.2 Execução dos experimentos e análise dos resultados

Uma vez identificada a melhor configuração de parâmetros para o algoritmo, considerando número de iterações, tamanho da população, tamanho do arquivador e método de arquivamento, foram executados conjuntos de experimentos a fim de comparar os resultados obtidos pelo algoritmo proposto com os resultados obtidos por algoritmos encontrados na literatura.

Na Fase 1, os resultados obtidos pelo EDA foram comparados aos resultados obtidos pelos algoritmos NSGA-III, NSGA-II e SPEA2. Na segunda fase dos experimentos, os algoritmos utilizados na comparação foram o NSGA-II e o SPEA2. O NSGA-III não foi considerado na Fase 2 dos experimentos por não estar implementado no *framework* de refatoração utilizado.

Todos os algoritmos foram executados utilizando o *framework* jMetal (DURILLO; NEBRO, 2011) e, para os algoritmos NSGA-II e SPEA2, foram utilizados os parâmetros padrão, definidos pelo *framework*; os parâmetros usados no NSGA-III foram apresentados em (DEB; JAIN, 2014). O NSGA-III foi escolhido por ser um algoritmo que está no estado-da-arte da pesquisa sobre otimização com muitos objetivos, o NSGA-II e o SPEA2 foram escolhidos porque são algoritmos evolucionários tradicionais e porque dois dos métodos de arquivamento investigados – o arquivador *Crowding Distance* e o arquivador SPEA2 – exploram procedimentos utilizados nestes algoritmos.

5.1.3 Indicador de qualidade e testes estatísticos utilizados

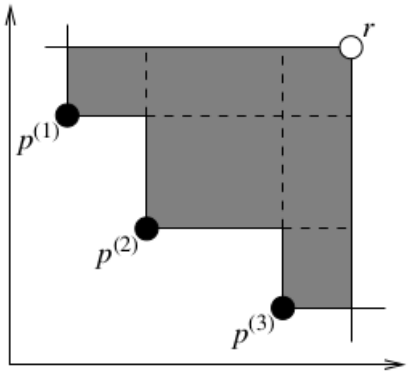
As soluções encontradas pelo *framework* de refatoração automática são avaliadas por meio de um conjunto de medidas de qualidade. Assim, ao comparar o desempenho dos algoritmos são observadas características como convergência (se os algoritmos conseguem otimizar as medidas de qualidade) e diversidade (se os algoritmos conseguem gerar soluções que abranjam diferentes combinações desses valores). Um conjunto de soluções ótimo é aquele com os melhores valores possíveis para as métricas de qualidade, para todas as medidas avaliadas.

Com o objetivo de avaliar a convergência e a diversidade do conjunto de soluções encontradas em relação à fronteira de Pareto, são utilizados indicadores de qualidade. Os indicadores de qualidade são definidos por Carvalho (2013) como sendo funções que mapeiam i conjuntos de soluções em um número real. Neste trabalho, o indicador de qualidade utilizado foi o hipervolume.

O hipervolume é uma métrica que determina a área coberta pela fronteira de Pareto aproximada, criada através da junção dos melhores valores para cada objetivo encontrado no experimento. É uma medida comum em trabalhos onde é feita a comparação entre algoritmos

multiobjetivos. Supondo um problema de minimização que lida com d objetivos, o hipervolume é a medida da região simultaneamente dominada por um conjunto $P = \{p^{(1)}, p^{(2)}, ..., p^{(n)}\}$ de n vetores de objetivos não dominados gerados em uma execução de um algoritmo que resolve um problema multiobjetivo e limitada por um ponto de referência r, tal que $r \geq (max_p p_1, ..., max_p p_d)$, onde $p = (p_1, ..., p_d) \in P$ e $max_p p_a$ é o maior valor atribuído a uma coordenada p_a, considerando o conjunto P.

Figura 11 – Exemplo do hipervolume em um problema com dois objetivos.



Fonte: Beume et al. (2009)

Na Figura 11, $p^{(1)}$, $p^{(2)}$ e $p^{(3)}$ são objetos não dominados gerados por algum algoritmo que resolve determinado problema multiobjetivo e r é um ponto de referência que tem coordenadas maiores ou iguais aos valores máximos que podem ser atribuídos às coordenadas de um ponto $p^{(i)}$. O hipervolume é a área da região coberta pela fronteira formada por $p^{(1)}$, $p^{(2)}$ e $p^{(3)}$, e limitada pelo ponto r.

Um maior hipervolume indica que o algoritmo conseguiu obter um conjunto de soluções mais convergente e diversificado, no contexto de SBSR, isso representa um conjunto de sequências de refatorações melhor. Nesse trabalho, não foi realizada uma análise de desempenho dos algoritmos considerando as métricas separadamente, foi analisado qual algoritmo obtém o melhor desempenho de maneira geral.

Tendo em vista que os algoritmos analisados são estocásticos, para que seja feita uma análise mais precisa, é necessário executar cada algoritmo diversas vezes e avaliar o seu desempenho com base no conjunto de valores de hipervolume obtidos. Com o objetivo de verificar se há diferença estatística entre os conjuntos de dados analisados em cada comparação, testes estatísticos foram aplicados considerando um valor de significância de 5%. O teste estatístico busca verificar se há diferença estatística entre os dados analisados. Após a análise do teste, é identificado qual algoritmo obteve os melhores resultados. Na Fase 1, foi utilizado o teste de Friedman, já na Fase 2, foi utilizado o teste Kruskal-Wallis. Para executar os testes, foi utilizada a ferramenta estatística R (R Core Team, 2012). Tanto na fase de validação do algoritmo quanto

na fase de aplicação do EDA no contexto de SBSR, os testes estatísticos foram utilizados apenas na comparação do algoritmo proposto com algoritmos encontrados na literatura.

5.2 Validação do EDA desenvolvido

A fim de verificar a adequabilidade dos EDA na resolução de problemas de otimização com muitos objetivos, decidiu-se testar o EDA em um problema de *benchmarking* e, para tal, o problema escolhido foi o Problema da Mochila Multiobjetivo (KIRLIK; SAYıN, 2014), onde deve-se decidir quais itens alocar em uma mochila, considerando os pesos e os custos de cada item, e respeitando o tamanho da mochila. Este problema foi escolhido por ser um problema largamente explorado na literatura e já existirem instâncias de teste nas quais são considerados muitos objetivos, permitindo assim que as técnicas de otimização com muitos objetivos a serem exploradas pudessem ser validadas com eficácia antes de serem aplicadas na busca por sequências de refatorações. As instâncias de teste encontradas consideram até cinco objetivos, sendo possível estender os casos de teste para situações onde um número maior de objetivos é considerado.

A versão do Problema da Mochila Multiobjetivo utilizada é definida em (KIRLIK; SAYıN, 2014) e considera uma capacidade $W > 0$ e n objetos. Cada objeto r é associado a um custo $w_r > 0$ e a um conjunto de p pesos v_r. A variável de decisão x_r indica se o objeto r foi adicionado à mochila ou não.

$$max \sum_{r=1}^{n} v_r^j x_r, j = \{1, ..., p\} \tag{5.1}$$

$$\sum_{r=1}^{n} w_r x_r \leq W \tag{5.2}$$

$$x_r \in \{0, 1\}, r = 1, ..., n \tag{5.3}$$

Na Equação 5.1 está representada a maximização das p funções objetivo, cada objetivo é representado pela soma dos pesos v_r^j dos objetos selecionados. A Equação 5.2 faz referência à restrição de capacidade, o custo total dos objetos selecionados deve ser menor ou igual a capacidade da mochila. A Equação 5.3 indica que as variáveis de decisão são binárias.

Sabendo que as variáveis de decisão do Problema da Mochila Multiobjetivo são binárias, foi utilizada uma variação do UMDA, cujos detalhes foram apresentados no Capítulo 4.

O EDA foi testado em algumas das instâncias do problema apresentadas em (KIRLIK, 2014). As instâncias diferem entre si no que diz respeito ao número de objetivos e ao número de itens considerados. As instâncias encontradas em (KIRLIK, 2014) consideram até cinco objetivos. Para testar o algoritmo proposto em instâncias com um número maior de objetivos, foram criadas novas instâncias, seguindo o mesmo procedimento descrito em (KIRLIK, 2014).

Foram considerados grupos de instâncias com 3, 5, 8 e 10 objetivos, cada grupo é composto por instâncias com 10, 20, 30, 40 e 50 itens. Foi considerada uma instância para cada número de objetivos e itens. Para cada algoritmo considerado na comparação dos resultados foram consideradas 30 execuções por instância do problema.

5.2.1 Configuração de parâmetros

Nesta seção são apresentadas as médias dos hipervolumes encontrados durante a configuração de parâmetros do EDA proposto, na fase de validação do algoritmo. Conforme apresentado no Capítulo 5, nesta fase dos experimentos o EDA foi utilizado para resolver o Problema da Mochila Multiobjetivo. Para encontrar a melhor configuração do algoritmo, foram investigados quatro métodos de arquivamento diferentes (*Crowding Distance*, Ideal, SPEA2 e MGA) e foram testados diferentes tamanhos para o arquivador e população utilizados pelo algoritmo. Os resultados obtidos para cada uma das configurações testadas são apresentados nas tabelas a seguir.

Tabela 9 – Problemas com 3 objetivos.

Itens	Pop. / Arq.	Crowding	Ideal	SPEA2	MGA
10	Pop. 100 / Arq. 5%	$7,89 \times 10^{10}$	$7,35 \times 10^{10}$	$7,57 \times 10^{10}$	$7,69 \times 10^{10}$
	Pop. 100 / Arq. 10%	$5,48 \times 10^{10}$	$6,74 \times 10^{10}$	$5,79 \times 10^{10}$	$5,81 \times 10^{10}$
	Pop. 100 / Arq. 20%	$1,71 \times 10^{10}$	$1,93 \times 10^{10}$	$2,03 \times 10^{10}$	$2,27 \times 10^{10}$
	Pop. 100 / Arq. 30%	$1,22 \times 10^{10}$	$1,12 \times 10^{10}$	$1,14 \times 10^{10}$	$8,64 \times 10^{9}$
	Pop. 100 / Arq. 40%	$8,58 \times 10^{9}$	$6,07 \times 10^{9}$	$8,03 \times 10^{9}$	$7,65 \times 10^{9}$
	Pop. 100 / Arq. 50%	$5,61 \times 10^{9}$	$5,56 \times 10^{9}$	$6,42 \times 10^{9}$	$5,29 \times 10^{9}$
	Pop. 200 / Arq. 5%	$7,40 \times 10^{10}$	$7,39 \times 10^{10}$	$7,53 \times 10^{10}$	$7,01 \times 10^{10}$
	Pop. 200 / Arq. 10%	$3,27 \times 10^{10}$	$3,34 \times 10^{10}$	$2,77 \times 10^{10}$	$3,07 \times 10^{10}$
	Pop. 200 / Arq. 20%	$1,20 \times 10^{10}$	$1,32 \times 10^{10}$	$1,26 \times 10^{10}$	$1,34 \times 10^{10}$
	Pop. 200 / Arq. 30%	$6,71 \times 10^{9}$	$6,90 \times 10^{9}$	$7,15 \times 10^{9}$	$7,10 \times 10^{9}$
	Pop. 200 / Arq. 40%	$5,09 \times 10^{9}$	$4,20 \times 10^{9}$	$4,51 \times 10^{9}$	$4,85 \times 10^{9}$
	Pop. 200 / Arq. 50%	$3,48 \times 10^{9}$	$3,86 \times 10^{9}$	$3,09 \times 10^{9}$	$4,05 \times 10^{9}$
20	Pop. 100 / Arq. 5%	$3,54 \times 10^{11}$	$3,50 \times 10^{11}$	$3,40 \times 10^{11}$	$3,15 \times 10^{11}$
	Pop. 100 / Arq. 10%	$2,42 \times 10^{11}$	$3,15 \times 10^{11}$	$2,63 \times 10^{11}$	$2,41 \times 10^{11}$
	Pop. 100 / Arq. 20%	$6,69 \times 10^{10}$	$5,45 \times 10^{10}$	$6,23 \times 10^{10}$	$4,56 \times 10^{10}$
	Pop. 100 / Arq. 30%	$2,85 \times 10^{10}$	$2,43 \times 10^{10}$	$2,40 \times 10^{10}$	$2,15 \times 10^{10}$
	Pop. 100 / Arq. 40%	$1,32 \times 10^{10}$	$1,66 \times 10^{10}$	$1,42 \times 10^{10}$	$1,38 \times 10^{10}$
	Pop. 100 / Arq. 50%	$1,05 \times 10^{10}$	$8,94 \times 10^{9}$	$1,07 \times 10^{10}$	$8,22 \times 10^{9}$
	Pop. 200 / Arq. 5%	$4,24 \times 10^{11}$	$3,47 \times 10^{11}$	$4,11 \times 10^{11}$	$3,99 \times 10^{11}$
	Pop. 200 / Arq. 10%	$7,93 \times 10^{10}$	$1,13 \times 10^{11}$	$1,32 \times 10^{11}$	$1,11 \times 10^{11}$
	Pop. 200 / Arq. 20%	$2,65 \times 10^{10}$	$2,84 \times 10^{10}$	$2,86 \times 10^{10}$	$2,22 \times 10^{10}$
	Pop. 200 / Arq. 30%	$1,18 \times 10^{10}$	$1,55 \times 10^{10}$	$1,66 \times 10^{10}$	$1,48 \times 10^{10}$
	Pop. 200 / Arq. 40%	$1,09 \times 10^{10}$	$9,72 \times 10^{9}$	$1,00 \times 10^{10}$	$9,36 \times 10^{9}$
	Pop. 200 / Arq. 50%	$8,50 \times 10^{9}$	$6,80 \times 10^{9}$	$6,73 \times 10^{9}$	$8,32 \times 10^{9}$
30	Pop. 100 / Arq. 5%	$1,03 \times 10^{12}$	$1,09 \times 10^{12}$	$1,23 \times 10^{12}$	$1,04 \times 10^{12}$
	Pop. 100 / Arq. 10%	$5,80 \times 10^{11}$	$4,61 \times 10^{11}$	$7,37 \times 10^{11}$	$3,71 \times 10^{11}$
	Pop. 100 / Arq. 20%	$1,14 \times 10^{11}$	$1,28 \times 10^{11}$	$1,21 \times 10^{11}$	$1,50 \times 10^{11}$
	Pop. 100 / Arq. 30%	$6,20 \times 10^{10}$	$8,19 \times 10^{10}$	$6,17 \times 10^{10}$	$7,03 \times 10^{10}$
	Pop. 100 / Arq. 40%	$4,80 \times 10^{10}$	$5,15 \times 10^{10}$	$3,97 \times 10^{10}$	$5,70 \times 10^{10}$
	Pop. 100 / Arq. 50%	$3,03 \times 10^{10}$	$3,77 \times 10^{10}$	$2,99 \times 10^{10}$	$2,80 \times 10^{10}$
	Pop. 200 / Arq. 5%	$8,54 \times 10^{11}$	$9,81 \times 10^{11}$	$1,03 \times 10^{12}$	$9,54 \times 10^{11}$
	Pop. 200 / Arq. 10%	$2,25 \times 10^{11}$	$1,55 \times 10^{11}$	$1,85 \times 10^{11}$	$2,03 \times 10^{11}$

	Pop. 200 / Arq. 20%	$8,80 \times 10^{10}$	$6,77 \times 10^{10}$	$7,91 \times 10^{10}$	$7,86 \times 10^{10}$
	Pop. 200 / Arq. 30%	$4,25 \times 10^{10}$	$3,90 \times 10^{10}$	$3,92 \times 10^{10}$	$3,75 \times 10^{10}$
	Pop. 200 / Arq. 40%	$2,32 \times 10^{10}$	$2,12 \times 10^{10}$	$2,45 \times 10^{10}$	$2,41 \times 10^{10}$
	Pop. 200 / Arq. 50%	$2,02 \times 10^{10}$	$2,20 \times 10^{10}$	$2,07 \times 10^{10}$	$1,86 \times 10^{10}$
40	Pop. 100 / Arq. 5%	$2,31 \times 10^{12}$	$2,27 \times 10^{12}$	$2,17 \times 10^{12}$	$2,05 \times 10^{12}$
	Pop. 100 / Arq. 10%	$1,01 \times 10^{12}$	$1,18 \times 10^{12}$	$1,25 \times 10^{12}$	$1,28 \times 10^{12}$
	Pop. 100 / Arq. 20%	$3,12 \times 10^{11}$	$2,43 \times 10^{11}$	$2,38 \times 10^{11}$	$2,34 \times 10^{11}$
	Pop. 100 / Arq. 30%	$1,16 \times 10^{11}$	$9,99 \times 10^{10}$	$1,10 \times 10^{11}$	$1,55 \times 10^{11}$
	Pop. 100 / Arq. 40%	$7,34 \times 10^{10}$	$7,72 \times 10^{10}$	$8,46 \times 10^{10}$	$8,29 \times 10^{10}$
	Pop. 100 / Arq. 50%	$5,24 \times 10^{10}$	$4,69 \times 10^{10}$	$5,80 \times 10^{10}$	$4,30 \times 10^{10}$
	Pop. 200 / Arq. 5%	$2,22 \times 10^{12}$	$2,10 \times 10^{12}$	$1,93 \times 10^{12}$	$2,08 \times 10^{12}$
	Pop. 200 / Arq. 10%	$3,96 \times 10^{11}$	$3,76 \times 10^{11}$	$3,21 \times 10^{11}$	$3,68 \times 10^{11}$
	Pop. 200 / Arq. 20%	$1,21 \times 10^{11}$	$1,38 \times 10^{11}$	$1,33 \times 10^{11}$	$1,18 \times 10^{11}$
	Pop. 200 / Arq. 30%	$7,57 \times 10^{10}$	$5,43 \times 10^{10}$	$7,02 \times 10^{10}$	$8,64 \times 10^{10}$
	Pop. 200 / Arq. 40%	$4,69 \times 10^{10}$	$4,21 \times 10^{10}$	$3,93 \times 10^{10}$	$4,73 \times 10^{10}$
	Pop. 200 / Arq. 50%	$3,06 \times 10^{10}$	$3,03 \times 10^{10}$	$3,18 \times 10^{10}$	$3,92 \times 10^{10}$
50	Pop. 100 / Arq. 5%	$2,49 \times 10^{12}$	$2,36 \times 10^{12}$	$3,33 \times 10^{12}$	$2,38 \times 10^{12}$
	Pop. 100 / Arq. 10%	$9,45 \times 10^{11}$	$6,57 \times 10^{11}$	$8,73 \times 10^{11}$	$5,30 \times 10^{11}$
	Pop. 100 / Arq. 20%	$2,67 \times 10^{11}$	$2,95 \times 10^{11}$	$2,26 \times 10^{11}$	$2,81 \times 10^{11}$
	Pop. 100 / Arq. 30%	$1,03 \times 10^{11}$	$1,21 \times 10^{11}$	$1,43 \times 10^{11}$	$1,23 \times 10^{11}$
	Pop. 100 / Arq. 40%	$7,86 \times 10^{10}$	$8,20 \times 10^{10}$	$1,02 \times 10^{11}$	$9,01 \times 10^{10}$
	Pop. 100 / Arq. 50%	$5,53 \times 10^{10}$	$5,69 \times 10^{10}$	$4,51 \times 10^{10}$	$3,85 \times 10^{10}$
	Pop. 200 / Arq. 5%	$1,49 \times 10^{12}$	$1,10 \times 10^{12}$	$1,25 \times 10^{12}$	$1,21 \times 10^{12}$
	Pop. 200 / Arq. 10%	$4,36 \times 10^{11}$	$4,31 \times 10^{11}$	$5,10 \times 10^{11}$	$4,71 \times 10^{11}$
	Pop. 200 / Arq. 20%	$1,26 \times 10^{11}$	$1,31 \times 10^{11}$	$1,20 \times 10^{11}$	$1,51 \times 10^{11}$
	Pop. 200 / Arq. 30%	$4,55 \times 10^{10}$	$7,91 \times 10^{10}$	$7,50 \times 10^{10}$	$5,28 \times 10^{10}$
	Pop. 200 / Arq. 40%	$4,29 \times 10^{10}$	$4,80 \times 10^{10}$	$3,53 \times 10^{10}$	$3,43 \times 10^{10}$
	Pop. 200 / Arq. 50%	$3,08 \times 10^{10}$	$3,20 \times 10^{10}$	$3,32 \times 10^{10}$	$3,47 \times 10^{10}$

Fonte: Produzido pelo autor.

Conforme apresentado na Tabela 9, ao testar o EDA em um problema com 3 objetivos, considerando as médias dos hipervolumes obtidos pelas diferentes configurações testadas, percebe-se que, na maioria dos casos, o algoritmo obtém os melhores resultados ao utilizar uma população composta por 100 indivíduos e um arquivador com capacidade para armazenar até 5% do tamanho da população. Apenas em um dos casos, aumentar o tamanho da população levou o algoritmo a obter um desempenho melhor. No que diz respeito aos métodos de arquivamento, os métodos *Crowding* e SPEA2 se destacaram nos problemas com 3 objetivos.

Tabela 10 – Problemas com 5 objetivos.

Itens	Pop. / Arq.	Crowding	Ideal	SPEA2	MGA
10	Pop. 100 / Arq. 5%	$1,05 \times 10^{18}$	$1,12 \times 10^{18}$	$1,08 \times 10^{18}$	$9,65 \times 10^{17}$
	Pop. 100 / Arq. 10%	$1,56 \times 10^{18}$	$1,65 \times 10^{18}$	$1,33 \times 10^{18}$	$1,30 \times 10^{18}$
	Pop. 100 / Arq. 20%	$1,58 \times 10^{18}$	$1,52 \times 10^{18}$	$1,65 \times 10^{18}$	$1,58 \times 10^{18}$
	Pop. 100 / Arq. 30%	$8,15 \times 10^{17}$	$7,45 \times 10^{17}$	$8,96 \times 10^{17}$	$8,32 \times 10^{17}$
	Pop. 100 / Arq. 40%	$3,83 \times 10^{17}$	$4,02 \times 10^{17}$	$3,94 \times 10^{17}$	$3,80 \times 10^{17}$
	Pop. 100 / Arq. 50%	$2,70 \times 10^{17}$	$2,52 \times 10^{17}$	$2,51 \times 10^{17}$	$2,33 \times 10^{17}$
	Pop. 200 / Arq. 5%	$1,88 \times 10^{18}$	$1,85 \times 10^{18}$	$1,43 \times 10^{18}$	$1,45 \times 10^{18}$
	Pop. 200 / Arq. 10%	$2,01 \times 10^{18}$	$1,95 \times 10^{18}$	$1,97 \times 10^{18}$	$1,98 \times 10^{18}$
	Pop. 200 / Arq. 20%	$5,49 \times 10^{17}$	$6,17 \times 10^{17}$	$5,67 \times 10^{17}$	$5,62 \times 10^{17}$
	Pop. 200 / Arq. 30%	$2,00 \times 10^{17}$	$2,34 \times 10^{17}$	$2,55 \times 10^{17}$	$1,99 \times 10^{17}$
	Pop. 200 / Arq. 40%	$7,45 \times 10^{16}$	$1,09 \times 10^{17}$	$1,34 \times 10^{17}$	$1,13 \times 10^{17}$

	Pop. 200 / Arq. 50%	$8,15 \times 10^{16}$	$4,78 \times 10^{16}$	$4,42 \times 10^{16}$	$4,00 \times 10^{16}$
20	Pop. 100 / Arq. 5%	$1,50 \times 10^{19}$	$1,91 \times 10^{19}$	$1,66 \times 10^{19}$	$1,25 \times 10^{19}$
	Pop. 100 / Arq. 10%	$2,57 \times 10^{19}$	$2,23 \times 10^{19}$	$2,13 \times 10^{19}$	$1,90 \times 10^{19}$
	Pop. 100 / Arq. 20%	$2,80 \times 10^{19}$	$2,37 \times 10^{19}$	$2,03 \times 10^{19}$	$2,11 \times 10^{19}$
	Pop. 100 / Arq. 30%	$2,15 \times 10^{19}$	$2,36 \times 10^{19}$	$1,94 \times 10^{19}$	$2,36 \times 10^{19}$
	Pop. 100 / Arq. 40%	$1,29 \times 10^{19}$	$1,91 \times 10^{19}$	$1,61 \times 10^{19}$	$1,38 \times 10^{19}$
	Pop. 100 / Arq. 50%	$1,06 \times 10^{19}$	$8,88 \times 10^{18}$	$7,21 \times 10^{18}$	$7,90 \times 10^{18}$
	Pop. 200 / Arq. 5%	$2,78 \times 10^{19}$	$2,43 \times 10^{19}$	$2,19 \times 10^{19}$	$2,14 \times 10^{19}$
	Pop. 200 / Arq. 10%	$2,81 \times 10^{19}$	$2,73 \times 10^{19}$	$2,63 \times 10^{19}$	$2,38 \times 10^{19}$
	Pop. 200 / Arq. 20%	$2,26 \times 10^{19}$	$2,28 \times 10^{19}$	$2,28 \times 10^{19}$	$2,40 \times 10^{19}$
	Pop. 200 / Arq. 30%	$9,78 \times 10^{18}$	$4,76 \times 10^{18}$	$4,29 \times 10^{18}$	$8,80 \times 10^{18}$
	Pop. 200 / Arq. 40%	$4,45 \times 10^{17}$	$1,61 \times 10^{18}$	$6,49 \times 10^{17}$	$3,30 \times 10^{18}$
	Pop. 200 / Arq. 50%	$2,98 \times 10^{17}$	$4,27 \times 10^{17}$	$2,64 \times 10^{17}$	$3,29 \times 10^{17}$
30	Pop. 100 / Arq. 5%	$1,59 \times 10^{20}$	$1,86 \times 10^{20}$	$1,61 \times 10^{20}$	$1,36 \times 10^{20}$
	Pop. 100 / Arq. 10%	$2,32 \times 10^{20}$	$2,11 \times 10^{20}$	$1,76 \times 10^{20}$	$1,84 \times 10^{20}$
	Pop. 100 / Arq. 20%	$1,17 \times 10^{20}$	$8,80 \times 10^{19}$	$1,46 \times 10^{20}$	$1,09 \times 10^{20}$
	Pop. 100 / Arq. 30%	$3,86 \times 10^{19}$	$3,35 \times 10^{19}$	$2,26 \times 10^{19}$	$3,21 \times 10^{19}$
	Pop. 100 / Arq. 40%	$2,66 \times 10^{18}$	$2,54 \times 10^{18}$	$2,35 \times 10^{18}$	$3,45 \times 10^{18}$
	Pop. 100 / Arq. 50%	$1,81 \times 10^{18}$	$2,02 \times 10^{18}$	$1,89 \times 10^{18}$	$1,12 \times 10^{18}$
	Pop. 200 / Arq. 5%	$2,62 \times 10^{20}$	$2,71 \times 10^{20}$	$2,24 \times 10^{20}$	$1,71 \times 10^{20}$
	Pop. 200 / Arq. 10%	$2,03 \times 10^{20}$	$2,09 \times 10^{20}$	$1,71 \times 10^{20}$	$1,60 \times 10^{20}$
	Pop. 200 / Arq. 20%	$1,89 \times 10^{19}$	$9,28 \times 10^{18}$	$2,32 \times 10^{19}$	$1,13 \times 10^{19}$
	Pop. 200 / Arq. 30%	$2,25 \times 10^{18}$	$2,56 \times 10^{18}$	$2,80 \times 10^{18}$	$2,62 \times 10^{18}$
	Pop. 200 / Arq. 40%	$1,69 \times 10^{18}$	$1,32 \times 10^{18}$	$1,54 \times 10^{18}$	$1,76 \times 10^{18}$
	Pop. 200 / Arq. 50%	$5,25 \times 10^{17}$	$1,22 \times 10^{18}$	$6,39 \times 10^{17}$	$6,83 \times 10^{17}$
40	Pop. 100 / Arq. 5%	$3,19 \times 10^{20}$	$3,89 \times 10^{20}$	$3,16 \times 10^{20}$	$2,54 \times 10^{20}$
	Pop. 100 / Arq. 10%	$5,66 \times 10^{20}$	$4,59 \times 10^{20}$	$3,93 \times 10^{20}$	$3,71 \times 10^{20}$
	Pop. 100 / Arq. 20%	$4,46 \times 10^{20}$	$2,86 \times 10^{20}$	$2,58 \times 10^{20}$	$3,75 \times 10^{20}$
	Pop. 100 / Arq. 30%	$2,73 \times 10^{20}$	$1,95 \times 10^{20}$	$1,44 \times 10^{20}$	$1,60 \times 10^{20}$
	Pop. 100 / Arq. 40%	$8,31 \times 10^{19}$	$5,42 \times 10^{19}$	$2,86 \times 10^{19}$	$4,52 \times 10^{19}$
	Pop. 100 / Arq. 50%	$5,70 \times 10^{18}$	$3,37 \times 10^{18}$	$7,80 \times 10^{18}$	$5,43 \times 10^{18}$
	Pop. 200 / Arq. 5%	$6,24 \times 10^{20}$	$5,07 \times 10^{20}$	$4,34 \times 10^{20}$	$3,98 \times 10^{20}$
	Pop. 200 / Arq. 10%	$5,29 \times 10^{20}$	$5,62 \times 10^{20}$	$4,43 \times 10^{20}$	$4,38 \times 10^{20}$
	Pop. 200 / Arq. 20%	$1,42 \times 10^{20}$	$1,18 \times 10^{20}$	$1,22 \times 10^{20}$	$1,29 \times 10^{20}$
	Pop. 200 / Arq. 30%	$1,27 \times 10^{19}$	$1,55 \times 10^{19}$	$7,70 \times 10^{18}$	$3,17 \times 10^{19}$
	Pop. 200 / Arq. 40%	$5,56 \times 10^{18}$	$4,23 \times 10^{18}$	$5,08 \times 10^{18}$	$5,09 \times 10^{18}$
	Pop. 200 / Arq. 50%	$2,60 \times 10^{18}$	$2,06 \times 10^{18}$	$1,43 \times 10^{18}$	$2,78 \times 10^{18}$
50	Pop. 100 / Arq. 5%	$1,28 \times 10^{21}$	$1,76 \times 10^{21}$	$1,55 \times 10^{21}$	$1,21 \times 10^{21}$
	Pop. 100 / Arq. 10%	$1,64 \times 10^{21}$	$1,34 \times 10^{21}$	$9,63 \times 10^{20}$	$7,97 \times 10^{20}$
	Pop. 100 / Arq. 20%	$1,59 \times 10^{20}$	$1,43 \times 10^{20}$	$1,33 \times 10^{20}$	$9,62 \times 10^{19}$
	Pop. 100 / Arq. 30%	$3,60 \times 10^{19}$	$3,67 \times 10^{19}$	$3,33 \times 10^{19}$	$3,44 \times 10^{19}$
	Pop. 100 / Arq. 40%	$1,30 \times 10^{19}$	$1,32 \times 10^{19}$	$1,46 \times 10^{19}$	$1,26 \times 10^{19}$
	Pop. 100 / Arq. 50%	$2,88 \times 10^{18}$	$6,06 \times 10^{18}$	$5,95 \times 10^{18}$	$8,12 \times 10^{18}$
	Pop. 200 / Arq. 5%	$2,43 \times 10^{21}$	$2,18 \times 10^{21}$	$1,95 \times 10^{21}$	$1,48 \times 10^{21}$
	Pop. 200 / Arq. 10%	$5,00 \times 10^{20}$	$2,41 \times 10^{20}$	$5,49 \times 10^{20}$	$2,26 \times 10^{20}$
	Pop. 200 / Arq. 20%	$2,72 \times 10^{19}$	$4,47 \times 10^{19}$	$5,04 \times 10^{19}$	$1,73 \times 10^{19}$
	Pop. 200 / Arq. 30%	$1,60 \times 10^{19}$	$9,71 \times 10^{18}$	$9,12 \times 10^{18}$	$7,30 \times 10^{18}$
	Pop. 200 / Arq. 40%	$3,59 \times 10^{18}$	$4,14 \times 10^{18}$	$3,05 \times 10^{18}$	$5,92 \times 10^{18}$
	Pop. 200 / Arq. 50%	$2,19 \times 10^{18}$	$2,34 \times 10^{18}$	$2,01 \times 10^{18}$	$2,20 \times 10^{18}$

Fonte: Produzido pelo autor.

De acordo com os dados apresentados na Tabela 10, ao testar o EDA em problemas onde são considerados 5 objetivos, diferentemente do que ocorreu ao testar o algoritmo em problemas com 3 objetivos, ao utilizar uma população maior o EDA teve um desempenho melhor. Em

todos os casos testados, os melhores resultados foram obtidos ao utilizar uma população com 200 indivíduos, enquanto a capacidade dos arquivadores variou entre 5% e 10% do tamanho da população. Nos problemas com 5 objetivos, o método de arquivamento *Crowding* se destacou, obtendo os melhores resultados na maioria dos casos testados.

Tabela 11 – Problemas com 8 objetivos.

Itens	Pop. / Arq.	Crowding	Ideal	SPEA2	MGA
10	Pop. 100 / Arq. 5%	$2,36 \times 10^{28}$	$4,69 \times 10^{28}$	$3,38 \times 10^{28}$	$2,16 \times 10^{28}$
	Pop. 100 / Arq. 10%	$5,28 \times 10^{28}$	$7,40 \times 10^{28}$	$6,64 \times 10^{28}$	$5,09 \times 10^{28}$
	Pop. 100 / Arq. 20%	$9,42 \times 10^{28}$	$9,35 \times 10^{28}$	$7,26 \times 10^{28}$	$6,32 \times 10^{28}$
	Pop. 100 / Arq. 30%	$9,70 \times 10^{28}$	$9,81 \times 10^{28}$	$9,91 \times 10^{28}$	$1,04 \times 10^{29}$
	Pop. 100 / Arq. 40%	$1,01 \times 10^{29}$	$1,09 \times 10^{29}$	$1,03 \times 10^{29}$	$9,99 \times 10^{28}$
	Pop. 100 / Arq. 50%	$9,21 \times 10^{28}$	$8,90 \times 10^{28}$	$9,71 \times 10^{28}$	$8,46 \times 10^{28}$
	Pop. 200 / Arq. 5%	$6,21 \times 10^{28}$	$8,73 \times 10^{28}$	$7,96 \times 10^{28}$	$5,62 \times 10^{28}$
	Pop. 200 / Arq. 10%	$1,09 \times 10^{29}$	$1,16 \times 10^{29}$	$8,38 \times 10^{28}$	$9,27 \times 10^{28}$
	Pop. 200 / Arq. 20%	$1,23 \times 10^{29}$	$1,21 \times 10^{29}$	$1,11 \times 10^{29}$	$1,27 \times 10^{29}$
	Pop. 200 / Arq. 30%	$9,35 \times 10^{28}$	$9,29 \times 10^{28}$	$9,89 \times 10^{28}$	$1,04 \times 10^{29}$
	Pop. 200 / Arq. 40%	$3,94 \times 10^{28}$	$3,48 \times 10^{28}$	$3,57 \times 10^{28}$	$3,86 \times 10^{28}$
	Pop. 200 / Arq. 50%	$1,90 \times 10^{28}$	$1,94 \times 10^{28}$	$1,63 \times 10^{28}$	$2,00 \times 10^{28}$
20	Pop. 100 / Arq. 5%	$1,01 \times 10^{31}$	$1,33 \times 10^{31}$	$1,13 \times 10^{31}$	$9,51 \times 10^{30}$
	Pop. 100 / Arq. 10%	$1,57 \times 10^{31}$	$1,85 \times 10^{31}$	$1,60 \times 10^{31}$	$1,23 \times 10^{31}$
	Pop. 100 / Arq. 20%	$2,67 \times 10^{31}$	$2,00 \times 10^{31}$	$1,93 \times 10^{31}$	$1,71 \times 10^{31}$
	Pop. 100 / Arq. 30%	$2,26 \times 10^{31}$	$1,46 \times 10^{31}$	$1,75 \times 10^{31}$	$1,35 \times 10^{31}$
	Pop. 100 / Arq. 40%	$1,37 \times 10^{31}$	$7,22 \times 10^{30}$	$1,56 \times 10^{31}$	$9,75 \times 10^{30}$
	Pop. 100 / Arq. 50%	$7,22 \times 10^{30}$	$4,57 \times 10^{30}$	$9,89 \times 10^{30}$	$2,96 \times 10^{30}$
	Pop. 200 / Arq. 5%	$1,76 \times 10^{31}$	$2,07 \times 10^{31}$	$1,81 \times 10^{31}$	$1,38 \times 10^{31}$
	Pop. 200 / Arq. 10%	$3,21 \times 10^{31}$	$2,18 \times 10^{31}$	$2,28 \times 10^{31}$	$1,88 \times 10^{31}$
	Pop. 200 / Arq. 20%	$2,49 \times 10^{31}$	$1,81 \times 10^{31}$	$2,50 \times 10^{31}$	$1,92 \times 10^{31}$
	Pop. 200 / Arq. 30%	$1,54 \times 10^{31}$	$8,84 \times 10^{30}$	$1,23 \times 10^{31}$	$9,15 \times 10^{30}$
	Pop. 200 / Arq. 40%	$3,32 \times 10^{30}$	$2,18 \times 10^{30}$	$1,44 \times 10^{30}$	$6,78 \times 10^{30}$
	Pop. 200 / Arq. 50%	$6,89 \times 10^{28}$	$1,84 \times 10^{29}$	$2,27 \times 10^{29}$	$9,77 \times 10^{29}$
30	Pop. 100 / Arq. 5%	$1,51 \times 10^{32}$	$2,67 \times 10^{32}$	$1,95 \times 10^{32}$	$1,35 \times 10^{32}$
	Pop. 100 / Arq. 10%	$2,98 \times 10^{32}$	$4,00 \times 10^{32}$	$3,19 \times 10^{32}$	$2,74 \times 10^{32}$
	Pop. 100 / Arq. 20%	$5,48 \times 10^{32}$	$4,56 \times 10^{32}$	$3,33 \times 10^{32}$	$2,89 \times 10^{32}$
	Pop. 100 / Arq. 30%	$5,09 \times 10^{32}$	$4,23 \times 10^{32}$	$3,50 \times 10^{32}$	$3,49 \times 10^{32}$
	Pop. 100 / Arq. 40%	$3,54 \times 10^{32}$	$2,45 \times 10^{32}$	$2,03 \times 10^{32}$	$1,70 \times 10^{32}$
	Pop. 100 / Arq. 50%	$8,56 \times 10^{31}$	$4,45 \times 10^{31}$	$1,13 \times 10^{32}$	$1,52 \times 10^{32}$
	Pop. 200 / Arq. 5%	$3,66 \times 10^{32}$	$4,77 \times 10^{32}$	$3,42 \times 10^{32}$	$3,00 \times 10^{32}$
	Pop. 200 / Arq. 10%	$6,38 \times 10^{32}$	$5,86 \times 10^{32}$	$3,78 \times 10^{32}$	$3,60 \times 10^{32}$
	Pop. 200 / Arq. 20%	$5,49 \times 10^{32}$	$4,46 \times 10^{32}$	$2,74 \times 10^{32}$	$4,12 \times 10^{32}$
	Pop. 200 / Arq. 30%	$2,18 \times 10^{32}$	$2,65 \times 10^{32}$	$8,22 \times 10^{31}$	$2,11 \times 10^{32}$
	Pop. 200 / Arq. 40%	$3,88 \times 10^{31}$	$7,07 \times 10^{31}$	$4,42 \times 10^{31}$	$8,36 \times 10^{29}$
	Pop. 200 / Arq. 50%	$1,47 \times 10^{29}$	$1,50 \times 10^{30}$	$9,12 \times 10^{29}$	$5,40 \times 10^{29}$
40	Pop. 100 / Arq. 5%	$7,36 \times 10^{32}$	$1,55 \times 10^{33}$	$1,10 \times 10^{33}$	$8,67 \times 10^{32}$
	Pop. 100 / Arq. 10%	$1,75 \times 10^{33}$	$2,13 \times 10^{33}$	$1,76 \times 10^{33}$	$1,36 \times 10^{33}$
	Pop. 100 / Arq. 20%	$3,00 \times 10^{33}$	$2,60 \times 10^{33}$	$1,94 \times 10^{33}$	$1,97 \times 10^{33}$
	Pop. 100 / Arq. 30%	$2,41 \times 10^{33}$	$1,98 \times 10^{33}$	$1,90 \times 10^{33}$	$1,15 \times 10^{33}$
	Pop. 100 / Arq. 40%	$1,48 \times 10^{33}$	$8,93 \times 10^{32}$	$1,01 \times 10^{33}$	$8,91 \times 10^{32}$
	Pop. 100 / Arq. 50%	$8,98 \times 10^{32}$	$5,46 \times 10^{32}$	$7,01 \times 10^{32}$	$2,73 \times 10^{32}$
	Pop. 200 / Arq. 5%	$1,98 \times 10^{33}$	$2,38 \times 10^{33}$	$1,70 \times 10^{33}$	$1,53 \times 10^{33}$
	Pop. 200 / Arq. 10%	$3,87 \times 10^{33}$	$2,96 \times 10^{33}$	$2,37 \times 10^{33}$	$2,00 \times 10^{33}$
	Pop. 200 / Arq. 20%	$2,68 \times 10^{33}$	$2,59 \times 10^{33}$	$2,24 \times 10^{33}$	$2,13 \times 10^{33}$
	Pop. 200 / Arq. 30%	$1,23 \times 10^{33}$	$1,09 \times 10^{33}$	$1,03 \times 10^{33}$	$7,04 \times 10^{32}$
	Pop. 200 / Arq. 40%	$1,85 \times 10^{32}$	$1,20 \times 10^{32}$	$1,33 \times 10^{32}$	$2,28 \times 10^{32}$

	Pop. 200 / Arq. 50%	$3,00 \times 10^{30}$	$1,14 \times 10^{32}$	$1,49 \times 10^{31}$	$1,21 \times 10^{30}$
50	Pop. 100 / Arq. 5%	$3,51 \times 10^{33}$	$7,22 \times 10^{33}$	$4,89 \times 10^{33}$	$4,21 \times 10^{33}$
	Pop. 100 / Arq. 10%	$8,79 \times 10^{33}$	$9,55 \times 10^{33}$	$8,23 \times 10^{33}$	$6,62 \times 10^{33}$
	Pop. 100 / Arq. 20%	$1,31 \times 10^{34}$	$9,42 \times 10^{33}$	$9,01 \times 10^{33}$	$8,38 \times 10^{33}$
	Pop. 100 / Arq. 30%	$1,04 \times 10^{34}$	$7,76 \times 10^{33}$	$8,03 \times 10^{33}$	$5,26 \times 10^{33}$
	Pop. 100 / Arq. 40%	$3,99 \times 10^{33}$	$3,39 \times 10^{33}$	$1,56 \times 10^{33}$	$2,87 \times 10^{33}$
	Pop. 100 / Arq. 50%	$1,28 \times 10^{33}$	$8,96 \times 10^{32}$	$5,54 \times 10^{32}$	$4,08 \times 10^{32}$
	Pop. 200 / Arq. 5%	$9,40 \times 10^{33}$	$1,16 \times 10^{34}$	$8,92 \times 10^{33}$	$6,72 \times 10^{33}$
	Pop. 200 / Arq. 10%	$1,57 \times 10^{34}$	$1,35 \times 10^{34}$	$1,13 \times 10^{34}$	$9,57 \times 10^{33}$
	Pop. 200 / Arq. 20%	$1,26 \times 10^{34}$	$9,31 \times 10^{33}$	$9,18 \times 10^{33}$	$7,49 \times 10^{33}$
	Pop. 200 / Arq. 30%	$8,17 \times 10^{32}$	$3,40 \times 10^{32}$	$2,54 \times 10^{33}$	$6,55 \times 10^{32}$
	Pop. 200 / Arq. 40%	$1,76 \times 10^{31}$	$2,12 \times 10^{31}$	$1,05 \times 10^{31}$	$1,26 \times 10^{31}$
	Pop. 200 / Arq. 50%	$6,23 \times 10^{30}$	$4,69 \times 10^{30}$	$7,05 \times 10^{30}$	$6,28 \times 10^{30}$

Fonte: Produzido pelo autor.

Conforme apresentado na Tabela 11, ao testar o EDA em problemas com 8 objetivos, considerando as médias dos hipervolumes obtidos pelas diferentes configurações testadas, percebe-se que, novamente, em todos os casos o algoritmo obteve melhor desempenho ao utilizar uma população composta por 200 indivíduos, enquanto, em 4 dos 5 casos testados, a capacidade de armazenamento do arquivador utilizado foi de até 10% do tamanho da população utilizada. Mais uma vez, o método de arquivamento *Crowding* se destacou entre os demais, obtendo os melhores resultados na maioria dos casos.

Tabela 12 – Problemas com 10 objetivos.

Itens	Pop. / Arq.	Crowding	Ideal	SPEA2	MGA
10	Pop. 100 / Arq. 5%	$5,98 \times 10^{35}$	$1,59 \times 10^{36}$	$1,17 \times 10^{36}$	$4,26 \times 10^{35}$
	Pop. 100 / Arq. 10%	$2,23 \times 10^{36}$	$3,11 \times 10^{36}$	$2,88 \times 10^{36}$	$1,20 \times 10^{36}$
	Pop. 100 / Arq. 20%	$5,03 \times 10^{36}$	$4,42 \times 10^{36}$	$4,60 \times 10^{36}$	$2,47 \times 10^{36}$
	Pop. 100 / Arq. 30%	$5,63 \times 10^{36}$	$4,28 \times 10^{36}$	$4,73 \times 10^{36}$	$4,12 \times 10^{36}$
	Pop. 100 / Arq. 40%	$4,82 \times 10^{36}$	$5,10 \times 10^{36}$	$4,11 \times 10^{36}$	$3,91 \times 10^{36}$
	Pop. 100 / Arq. 50%	$4,03 \times 10^{36}$	$5,12 \times 10^{36}$	$4,98 \times 10^{36}$	$4,84 \times 10^{36}$
	Pop. 200 / Arq. 5%	$2,74 \times 10^{36}$	$4,65 \times 10^{36}$	$2,34 \times 10^{36}$	$1,71 \times 10^{36}$
	Pop. 200 / Arq. 10%	$6,55 \times 10^{36}$	$6,98 \times 10^{36}$	$5,16 \times 10^{36}$	$3,70 \times 10^{36}$
	Pop. 200 / Arq. 20%	$6,58 \times 10^{36}$	$6,32 \times 10^{36}$	$6,71 \times 10^{36}$	$5,69 \times 10^{36}$
	Pop. 200 / Arq. 30%	$6,22 \times 10^{36}$	$6,28 \times 10^{36}$	$6,06 \times 10^{36}$	$6,25 \times 10^{36}$
	Pop. 200 / Arq. 40%	$2,26 \times 10^{36}$	$1,84 \times 10^{36}$	$1,98 \times 10^{36}$	$2,64 \times 10^{36}$
	Pop. 200 / Arq. 50%	$9,33 \times 10^{35}$	$9,56 \times 10^{35}$	$9,13 \times 10^{35}$	$6,16 \times 10^{35}$
20	Pop. 100 / Arq. 5%	$3,20 \times 10^{38}$	$6,33 \times 10^{38}$	$4,16 \times 10^{38}$	$2,99 \times 10^{38}$
	Pop. 100 / Arq. 10%	$7,48 \times 10^{38}$	$9,27 \times 10^{38}$	$9,83 \times 10^{38}$	$6,16 \times 10^{38}$
	Pop. 100 / Arq. 20%	$1,56 \times 10^{39}$	$1,37 \times 10^{39}$	$1,14 \times 10^{39}$	$8,09 \times 10^{38}$
	Pop. 100 / Arq. 30%	$1,51 \times 10^{39}$	$1,47 \times 10^{39}$	$1,25 \times 10^{39}$	$9,60 \times 10^{38}$
	Pop. 100 / Arq. 40%	$1,52 \times 10^{39}$	$1,49 \times 10^{39}$	$1,35 \times 10^{39}$	$1,04 \times 10^{39}$
	Pop. 100 / Arq. 50%	$1,22 \times 10^{39}$	$9,56 \times 10^{38}$	$1,19 \times 10^{39}$	$1,06 \times 10^{39}$
	Pop. 200 / Arq. 5%	$8,04 \times 10^{38}$	$1,10 \times 10^{39}$	$1,05 \times 10^{39}$	$5,84 \times 10^{38}$
	Pop. 200 / Arq. 10%	$1,78 \times 10^{39}$	$1,50 \times 10^{39}$	$1,30 \times 10^{39}$	$9,38 \times 10^{38}$
	Pop. 200 / Arq. 20%	$1,94 \times 10^{39}$	$1,73 \times 10^{39}$	$1,53 \times 10^{39}$	$1,02 \times 10^{39}$
	Pop. 200 / Arq. 30%	$1,50 \times 10^{39}$	$1,82 \times 10^{39}$	$1,74 \times 10^{39}$	$1,17 \times 10^{39}$
	Pop. 200 / Arq. 40%	$1,48 \times 10^{39}$	$1,07 \times 10^{39}$	$1,09 \times 10^{39}$	$8,91 \times 10^{38}$
	Pop. 200 / Arq. 50%	$7,22 \times 10^{38}$	$4,20 \times 10^{38}$	$7,06 \times 10^{38}$	$6,70 \times 10^{38}$
30	Pop. 100 / Arq. 5%	$6,28 \times 10^{39}$	$1,67 \times 10^{40}$	$9,40 \times 10^{39}$	$7,32 \times 10^{39}$
	Pop. 100 / Arq. 10%	$2,09 \times 10^{40}$	$2,71 \times 10^{40}$	$2,08 \times 10^{40}$	$1,44 \times 10^{40}$
	Pop. 100 / Arq. 20%	$4,07 \times 10^{40}$	$3,40 \times 10^{40}$	$2,51 \times 10^{40}$	$2,15 \times 10^{40}$
	Pop. 100 / Arq. 30%	$4,43 \times 10^{40}$	$3,65 \times 10^{40}$	$2,78 \times 10^{40}$	$2,40 \times 10^{40}$

	Pop. 100 / Arq. 40%	$3,37 \times 10^{40}$	$2,86 \times 10^{40}$	$2,80 \times 10^{40}$	$1,83 \times 10^{40}$
	Pop. 100 / Arq. 50%	$2,70 \times 10^{40}$	$2,70 \times 10^{40}$	$2,43 \times 10^{40}$	$1,41 \times 10^{40}$
	Pop. 200 / Arq. 5%	$2,24 \times 10^{40}$	$2,95 \times 10^{40}$	$2,16 \times 10^{40}$	$1,51 \times 10^{40}$
	Pop. 200 / Arq. 10%	$5,23 \times 10^{40}$	$4,12 \times 10^{40}$	$3,10 \times 10^{40}$	$2,34 \times 10^{40}$
	Pop. 200 / Arq. 20%	$4,87 \times 10^{40}$	$4,79 \times 10^{40}$	$3,66 \times 10^{40}$	$2,71 \times 10^{40}$
	Pop. 200 / Arq. 30%	$4,40 \times 10^{40}$	$3,58 \times 10^{40}$	$2,99 \times 10^{40}$	$2,05 \times 10^{40}$
	Pop. 200 / Arq. 40%	$1,84 \times 10^{40}$	$2,28 \times 10^{40}$	$1,36 \times 10^{40}$	$1,50 \times 10^{40}$
	Pop. 200 / Arq. 50%	$6,49 \times 10^{38}$	$5,58 \times 10^{39}$	$3,13 \times 10^{39}$	$2,78 \times 10^{39}$
40	Pop. 100 / Arq. 5%	$4,83 \times 10^{41}$	$8,68 \times 10^{41}$	$6,36 \times 10^{41}$	$4,96 \times 10^{41}$
	Pop. 100 / Arq. 10%	$1,04 \times 10^{42}$	$1,40 \times 10^{42}$	$1,11 \times 10^{42}$	$6,85 \times 10^{41}$
	Pop. 100 / Arq. 20%	$1,80 \times 10^{42}$	$1,10 \times 10^{42}$	$1,08 \times 10^{42}$	$7,66 \times 10^{41}$
	Pop. 100 / Arq. 30%	$1,00 \times 10^{42}$	$8,92 \times 10^{41}$	$7,39 \times 10^{41}$	$4,59 \times 10^{41}$
	Pop. 100 / Arq. 40%	$1,76 \times 10^{41}$	$2,57 \times 10^{41}$	$2,02 \times 10^{41}$	$2,80 \times 10^{39}$
	Pop. 100 / Arq. 50%	$6,03 \times 10^{40}$	$6,18 \times 10^{40}$	$1,40 \times 10^{39}$	$7,16 \times 10^{40}$
	Pop. 200 / Arq. 5%	$1,22 \times 10^{42}$	$1,55 \times 10^{42}$	$1,29 \times 10^{42}$	$1,15 \times 10^{42}$
	Pop. 200 / Arq. 10%	$2,17 \times 10^{42}$	$1,95 \times 10^{42}$	$1,27 \times 10^{42}$	$1,08 \times 10^{42}$
	Pop. 200 / Arq. 20%	$1,35 \times 10^{42}$	$9,17 \times 10^{41}$	$8,93 \times 10^{41}$	$9,76 \times 10^{41}$
	Pop. 200 / Arq. 30%	$9,46 \times 10^{39}$	$1,86 \times 10^{41}$	$2,01 \times 10^{41}$	$1,47 \times 10^{41}$
	Pop. 200 / Arq. 40%	$2,83 \times 10^{38}$	$5,67 \times 10^{38}$	$5,08 \times 10^{38}$	$2,05 \times 10^{39}$
	Pop. 200 / Arq. 50%	$1,70 \times 10^{38}$	$2,17 \times 10^{38}$	$1,65 \times 10^{38}$	$3,39 \times 10^{37}$
50	Pop. 100 / Arq. 5%	$6,30 \times 10^{41}$	$1,76 \times 10^{42}$	$1,07 \times 10^{42}$	$6,94 \times 10^{41}$
	Pop. 100 / Arq. 10%	$1,93 \times 10^{42}$	$2,79 \times 10^{42}$	$2,07 \times 10^{42}$	$1,50 \times 10^{42}$
	Pop. 100 / Arq. 20%	$4,66 \times 10^{42}$	$3,60 \times 10^{42}$	$3,04 \times 10^{42}$	$2,45 \times 10^{42}$
	Pop. 100 / Arq. 30%	$5,78 \times 10^{42}$	$3,68 \times 10^{42}$	$3,15 \times 10^{42}$	$2,32 \times 10^{42}$
	Pop. 100 / Arq. 40%	$3,85 \times 10^{42}$	$2,37 \times 10^{42}$	$2,68 \times 10^{42}$	$1,53 \times 10^{42}$
	Pop. 100 / Arq. 50%	$2,39 \times 10^{42}$	$1,72 \times 10^{42}$	$1,14 \times 10^{42}$	$9,89 \times 10^{41}$
	Pop. 200 / Arq. 5%	$1,99 \times 10^{42}$	$3,26 \times 10^{42}$	$2,37 \times 10^{42}$	$1,60 \times 10^{42}$
	Pop. 200 / Arq. 10%	$5,57 \times 10^{42}$	$4,20 \times 10^{42}$	$3,51 \times 10^{42}$	$2,68 \times 10^{42}$
	Pop. 200 / Arq. 20%	$6,68 \times 10^{42}$	$4,66 \times 10^{42}$	$4,89 \times 10^{42}$	$2,64 \times 10^{42}$
	Pop. 200 / Arq. 30%	$5,49 \times 10^{42}$	$3,75 \times 10^{42}$	$3,51 \times 10^{42}$	$2,62 \times 10^{42}$
	Pop. 200 / Arq. 40%	$1,98 \times 10^{42}$	$1,06 \times 10^{42}$	$7,70 \times 10^{41}$	$7,72 \times 10^{41}$
	Pop. 200 / Arq. 50%	$2,76 \times 10^{41}$	$5,94 \times 10^{39}$	$2,45 \times 10^{41}$	$5,60 \times 10^{39}$

Fonte: Produzido pelo autor.

Ao analisar os resultados apresentados na Tabela 12, percebe-se que ao aumentar o número de objetivos para 10, o EDA, de forma similar ao que ocorreu nos problemas com 8 objetivos, obtém melhores resultados ao utilizar uma população composta por 200 indivíduos e, na maioria dos casos, um arquivador com capacidade para armazenar até 10% do tamanho da população utilizada. Com relação ao método de arquivamento utilizado, mais uma vez, assim como ocorreu nos problemas com 5 e 8 objetivos, o método *Crowding* obteve destaque, conseguindo os melhores resultados na maioria dos casos.

Após a execução de experimentos testando as diferentes combinações de parâmetros apresentadas nas Tabelas 9, 10, 11 e 12, percebeu-se que os melhores resultados foram obtidos ao utilizar arquivadores cujo tamanho variavam entre 5% e 10% da população. Ao analisar o tamanho da população utilizada, nota-se que a medida que o número de objetivos do problema aumenta, as configurações onde foram utilizadas populações compostas por 200 indivíduos se destacam. Entretanto, é possível perceber que bons resultados também podem ser obtidos ao realizar um *trade-off* entre o tamanho da população e o tamanho do arquivador utilizado.

Uma vez realizada a análise dos parâmetros, os resultados obtidos pelo EDA foram

comparados aos resultados obtidos por algoritmos encontrados na literatura. Na seção seguinte são apresentados detalhes sobre esta etapa.

5.2.2 Resultados e análise

Na Tabela 13 é mostrado o resumo da comparação entre todos os hipervolumes obtidos pelo EDA considerando uma população de 200 indivíduos e um arquivador com capacidade para até 10% do tamanho da população – esta configuração foi escolhida por se destacar entre as demais configurações analisadas, nos problemas onde é considerado um número maior de objetivos – e os resultados obtidos pelos algoritmos SPEA2, NSGA-II e NSGA-III. Em cada célula são apresentados os algoritmos que obtiveram melhores resultados, de acordo com o teste estatístico. Observando a tabela, percebe-se que o EDA obteve os melhores resultados, de acordo com o teste de Friedman, nas instâncias com 8 e 10 objetivos, independente da quantidade de itens considerados na instância. O NSGA-III obteve um resultado semelhante na instância onde são considerados 10 objetivos, apenas quando 10 itens são considerados; os algoritmos SPEA2 e NSGA-II não obtiveram bons resultados nestes cenários.

Tabela 13 – Comparação entre os algoritmos (hipervolumes).

# Itens# Obj	3	5	8	10
10	NSGA-III, NSGA-II e SPEA2	EDA (Crowding, Ideal, SPEA2 e MGA) e NSGA-III	EDA (Crowding, Ideal e MGA)	EDA (Crowding, Ideal e SPEA2) e NSGA-III
20	NSGA-III, NSGA-II e SPEA2	EDA (Crowding, Ideal, SPEA2 e MGA) e NSGA-III	EDA (Crowding e SPEA2)	EDA (Crowding, Ideal e SPEA2)
30	NSGA-III, NSGA-II e SPEA2	Todos os algoritmos	EDA (Crowding e Ideal)	EDA (Crowding e Ideal)
40	NSGA-II e SPEA2	EDA (Crowding, Ideal e SPEA2) e NSGA-III	EDA (Crowding e Ideal)	EDA (Crowding, Ideal e SPEA2)
50	NSGA-III, NSGA-II e SPEA2	NSGA-II e SPEA2	EDA (Crowding e Ideal)	EDA (Crowding e Ideal)

Fonte: Produzido pelo autor.

Todos os algoritmos considerados no experimento obtiveram um desempenho competitivo no problema com 5 objetivos. Todos os algoritmos conseguiram bons resultados em pelo menos uma instância. O EDA e o NSGA-III conseguiram resultados similares nas instâncias com 10, 20, 30 e 40 itens. Os algoritmos tradicionais, SPEA2 e NSGA-II, superaram os outros algoritmos na instância com 50 itens e também conseguiram bons resultados na instância com 30 itens. Por fim, o SPEA2, o NSGA-II e o NSGA-III superaram o EDA nas instâncias onde foram considerados apenas três objetivos.

Em relação ao método de arquivamento, é perceptível que a escolha do método não é um fator que impacta na performance do EDA de forma crítica. De maneira geral, todos os métodos testados obtiveram uma boa performance e, na maioria dos casos, os arquivadores *Crowding Distance*, Ideal e SPEA2 obtiveram resultados similares. Entretanto, nas instâncias com 10 objetivos, os arquivadores *Crowding Distance* e Ideal sempre conseguiram bons resultados.

Observando os resultados obtidos na primeira fase dos experimentos, percebe-se que o EDA proposto tem potencial para ser aplicado em problemas com muitos objetivos discretos. Os bons resultados indicaram a possibilidade de aplicar o EDA no contexto da SBSR, detalhes sobre esta fase do trabalho são apresentados na seção a seguir.

5.3 Aplicação do EDA proposto no contexto de SBSR

Após a fase de validação do algoritmo proposto em um problema de *benchmarking*, o EDA foi aplicado ao problema de busca por sequências de refatorações. Os softwares utilizados nesta fase dos experimentos foram os mesmos utilizados por (BEZERRA, 2014): um Software Controle (SC) e dois softwares *open source*. Os softwares *open source* escolhidos foram o Beaver[1] – um gerador de parsers LALR – e o Gantt[2] 1.1.02 – software utilizado no planejamento de projetos de software. O autor justifica a escolha destes softwares baseando-se no fato de serem *open source* e de serem desenvolvidos utilizando a linguagem JAVA. Além disso, na escolha dos softwares, foi considerada a quantidade de classes que cada aplicação possui, fator a ser considerado, uma vez que o *framework*, detalhado no Capítulo 2 e utilizado na condução dos experimentos, não encontra-se otimizado. Mais detalhes sobre os softwares são apresentados na Tabela 14.

Tabela 14 – Softwares utilizados nos experimentos

Software	Num. Linhas de Código	Num. Classes	Refatorações Avaliadas	Num. Refatorações Encontradas
SC	155	10	135	32
Beaver	6008	89	1158	247
Gantt1102	21222	296	8040	1433

Fonte: Adaptado de (BEZERRA, 2014)

Conforme dito anteriormente, no problema de busca por sequências de refatorações, métricas da Engenharia de Software são mapeadas como funções objetivo que serão utilizadas para guiar o processo de busca. Neste trabalho, as métricas do modelo QMOOD (detalhado no Capítulo 2) foram utilizadas nesta fase dos experimentos. O conjunto de atributos de qualidade considerado foi o seguinte: Flexibilidade, Entendimento, Reusabilidade, Funcionalidade, Extensibilidade e Efetividade.

5.3.1 Definindo a quantidade de avaliações de função objetivo

Uma vez definidos os softwares e as métricas que seriam utilizadas, foi preciso definir o número de avaliações de função objetivo consideradas ao aplicar o EDA no problema de busca por sequências de refatorações. Esta etapa é necessária pelo fato de uma execução do algoritmo neste problema demandar um tempo muito maior do que uma execução no problema de

[1] http://beaver.sourceforge.net/
[2] http://www.ganttproject.biz/

benchmarking. Utilizar um número grande de avaliações de função objetivo seria muito custoso devido ao número de combinações de parâmetros a serem testadas em diferentes softwares. Diante deste cenário, é proibitiva a exploração de um número grande de avaliações de função objetivo nesta fase dos experimentos.

Para definir o número de avaliações de função objetivo, foi realizada uma execução do algoritmo onde o limite de avaliações foi definido como 10.000. Durante a execução, a cada 1.000 avaliações de função objetivo era calculado o hipervolume da população atual. A partir disso, foi possível perceber até que ponto o aumento da quantidade de avaliações impacta no desempenho do EDA. A execução foi realizada utilizando o software Beaver e as métricas do modelo QMOOD foram mapeadas como funções objetivo. Após o término da execução, percebeu-se que antes das 1.000 avaliações de função objetivo o desempenho do EDA não sofre melhorias significativas. Diante disso, decidiu-se considerar, nesta fase dos experimentos, execuções onde fossem realizadas 2.000 avaliações de função objetivo.

5.3.2 Comparação dos métodos de arquivamento

Após a definição da quantidade de avaliações de função objetivo, foi necessário especificar o método de arquivamento e o tamanho do arquivador utilizado pelo EDA. Para isso, utilizando o Software Beaver, foram comparados os resultados obtidos após a execução do algoritmo, utilizando os arquivadores *Crowding* e Ideal com a capacidade do arquivador variando entre 10% e 30% do tamanho da população. Novamente, o critério de comparação foi a média hipervolumes obtidos após 10 execuções realizadas para cada uma das diferentes configurações. Os resultados obtidos são apresentados na Tabela 15.

Tabela 15 – Comparação dos métodos de arquivamento (Média e Desvio Padrão dos hipervolumes).

	Crowding			Ideal		
Arquivador	10%	20%	30%	10%	20%	30%
Média	1,0749886	1,0938887	1,0992106	1,1569353	1,0952409	1,1664133
Desvio Padrão	0,0019978965	0,0060870651	0,0039025554	0,0227648027	0,0050854254	0,0271346451

Fonte: Produzido pelo autor.

Conforme apresentado na Tabela 15, ao utilizar o arquivador Ideal, com capacidade de até 30% do tamanho da população, o algoritmo proposto obteve melhores resultados. Por isso, essa configuração foi adotada para a realização das próximas etapas dos experimentos.

5.3.3 Comparação dos modelos probabilísticos

Uma vez definido o modelo probabilístico a ser utilizado, os modelos probabilísticos apresentados na Seção 4.2 foram comparados. A comparação entre os dois modelos foi conduzida da seguinte forma: O software utilizado nesta etapa foi o SC e, novamente as métricas do modelo QMOOD foram mapeadas como funções objetivo. O critério de comparação foi a média

hipervolumes obtidos após 10 execuções realizadas para cada um dos modelos probabilísticos. O método de arquivamento utilizado pelo EDA nesta etapa foi o *Crowding*, com um arquivador cuja capacidade foi definida como 30% da população. Os resultados obtidos são apresentados na Tabela 16.

Tabela 16 – Comparação dos modelos probabilísticos (Média e Desvio Padrão dos hipervolumes).

	UMDA	EHM
Média	5,4727146	3,5109829
Desvio Padrão	0,1857018328	0,0552738871

Fonte: Produzido pelo autor.

Diante dos resultados obtidos após a comparação dos dois modelos probabilísticos, percebeu-se a superioridade do UMDA. Considerando isto, decidiu-se utilizar o UMDA nas etapas seguintes dos experimentos.

5.3.4 Comparando o EDA com outros algoritmos da literatura

Após a configuração dos parâmetros do EDA, o seu desempenho foi comparado ao desempenho de algoritmos encontrados na literatura. Os algoritmos escolhidos foram o NSGA-II e o SPEA2, ambos implementados no jMetal. Para comparar os algoritmos, foram utilizados o Software Controle, o Beaver e o Gantt. Nesta comparação, foram consideradas sequências de refatorações compostas por 10, 20 e 30 refatorações quando os testes foram realizados no Beaver e no Gantt. Ao testar os algoritmos no Software Controle foram consideradas apenas sequências de 10 refatorações, esta restrição foi adicionada por causa do número de refatorações válidas encontradas no SC. No cálculo da média dos hipervolumes e na execução do teste estatístico, para cada tamanho de solução, foram consideradas 10 execuções por algoritmo. Os resultados obtidos ao testar os algoritmos no SC são apresentados na Tabela 17.

Tabela 17 – Comparação entre o EDA e algoritmos da literatura – Software Controle (Média e Desvio Padrão dos hipervolumes).

	EDA		SPEA2		NSGA-II	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Sequência Tam. 10	5,4727146	0,1857018328	4,9176718	0,0878623523	4,8775239	0,0696236864

Fonte: Produzido pelo autor.

Ao verificar os resultados apresentados na Tabela 17, percebe-se que, ao comparar as médias dos hipervolumes obtidos, o EDA supera tanto o SPEA2 quanto o NSGA-II. Nesta comparação, o método de arquivamento utilizado pelo EDA foi o *crowding*. Ao realizar o teste de Kruskall-Wallis, constatou-se que não existe diferença estatística entre os três conjuntos de dados analisados.

Após testar os algoritmos no SC, os algoritmos foram testados no Beaver. Os resultados obtidos são apresentados na Tabela 18.

Tabela 18 – Comparação entre o EDA e algoritmos da literatura – Beaver (Média e Desvio Padrão dos hipervolumes).

	EDA		SPEA2		NSGA-II	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Sequência Tam. 10	1,1664133	0,027134645	1,2025856	0,0227655056	1,259949	0,0281472558
Sequência Tam. 20	1,1605741	0,020803803	1,2164798	0,0229512296	1,1315141	0,0046511384
Sequência Tam. 30	1,1658556	0,023110716	1,2366202	0,0219744738	1,2138618	0,0278391128

Fonte: Produzido pelo autor.

Ao analisar os dados, percebe-se que as médias dos hipervolumes obtidos pelo EDA não sofrem mudanças significativas ao variar o tamanho da sequência de refatorações. Além disso, é perceptível que o NSGA-II obteve um desempenho superior ao dos outros algoritmos ao ser testado no Beaver, quando consideradas sequências compostas por 10 refatorações. Quando sequências compostas por 20 refatorações são utilizadas, o EDA obteve um desempenho superior ao do NSGA-II, entretanto, é superado pelo SPEA2. Por fim, ao considerar sequências de 30 refatorações, o SPEA2 superou tanto o EDA quanto o NSGA-II. Ao executar o teste de Kruskal-Wallis considerando um valor de significância de 5%, constatou-se que existe diferença estatística entre todos os conjuntos de dados obtidos nesta etapa dos experimentos, exceto entre os valores dos hipervolumes obtidos pelo EDA e pelo NSGA-II, quando consideradas sequências compostas por 30 refatorações.

Por fim, foram realizados testes no software Gantt. Novamente, foram consideradas sequências compostas por 10, 20 e 30 refatorações. Os resultados obtidos nesta etapa dos experimentos são apresentados na Tabela 19.

Tabela 19 – Comparação entre o EDA e algoritmos da literatura – Gantt (Média e Desvio Padrão dos hipervolumes).

	EDA		SPEA2		NSGA-II	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Sequência Tam. 10	1,0195304	0,00147641	1,0813146	0,0073378592	1,104848	0,00873014
Sequência Tam. 20	1,0263121	0,0023386	1,1111187	0,0088159476	1,079471	0,00461677
Sequência Tam. 30	1,0623689	0,00911645	1,1832531	0,0087122744	1,13423	0,00992574

Conforme apresentado na Tabela 19, assim como ocorreu ao testar o EDA no Beaver, as médias dos hipervolumes não sofrem mudanças consideráveis ao variar o tamanho da solução. Comparando as médias dos hipervolumes obtidos pelos três algoritmos, percebe-se que, quando consideradas sequências de 10 refatorações, o NSGA-II obtém um melhor resultado. Ao aumentar o tamanho das sequências de refatorações para 20, o SPEA2 supera os outros algoritmos. Por fim, quando consideradas sequências compostas por 30 refatorações, novamente o SPEA2 supera o EDA e o NSGA-II. Além da comparação das médias dos hipervolumes, foi executado o teste de Kruskal-Wallis considerando um valor de significância de 5%. O teste apontou uma diferença estatística entre os valores dos hipervolumes obtidos pelo EDA quando comparados com os

valores obtidos pelos outros algoritmos, exceto quando os valores obtidos pelo EDA quando consideradas sequências compostas por 30 refatorações são comparados aos valores obtidos pelo SPEA2. Além disso, existe diferença estatística entre os valores dos hipervolumes obtidos pelo SPEA2 e pelo NSGA-II.

Diante dos resultados obtidos ao aplicar o EDA no contexto de SBSR percebe-se que o algoritmo proposto não conseguiu superar, em todos os cenários, os algoritmos da literatura aos quais foi comparado, superando o NSGA-II e o SPEA quando testado no SC, e o NSGA-II quando testado no Beaver considerando sequências compostas por 20 refatorações. O resultado obtido na Fase 2 dos experimentos é contrário ao resultado obtido na primeira fase, onde o EDA foi testado no Problema da Mochila Multiobjetivo e obteve um desempenho superior ao dos demais algoritmos na maioria dos cenários testados, sobretudo quando considerados mais objetivos no problema.

Um fator que muda significativamente nos dois cenários onde o EDA foi testado é a quantidade de valores que as variáveis de uma solução pode assumir. No problema da mochila, as variáveis de decisão são binárias, o universo de valores possíveis para uma variável se limita à 0 ou 1; nesse cenário, o EDA, combinado aos métodos de arquivamento, obteve sucesso e superou os algoritmos da literatura. Em (PAUL; IBA, 2002), o UMDA obtém bons resultados ao resolver o Problema das *n* rainhas, onde a solução é uma permutação de números inteiros e é considerado um universo de até 15 valores possíveis para uma variável de decisão. No contexto de SBSR, este universo de valores cresce consideravelmente, no Beaver são consideradas 247 refatorações e no Gantt são consideradas 1.433 refatorações possíveis. Considerando isto, e analisando os resultados obtidos nas duas fases dos experimentos, percebe-se que ao aumentar o tamanho do universo de valores possíveis para uma variável de decisão, mesmo utilizando métodos de arquivamento, o modelo probabilístico não consegue evoluir na busca por soluções melhores, não sendo capaz de superar os algoritmos da literatura. Além do UMDA, foi testado outro modelo probabilístico – originalmente utilizado para resolver o Problema do Caixeiro Viajante (cuja estrutura da solução é semelhante às sequências de refatorações) –, entretanto, os resultados foram piores. Apesar disso, a extensão do algoritmo para que se alcance melhores resultados é algo possível de ser realizado.

6
Conclusão

O objetivo principal desse trabalho foi investigar o desempenho de algoritmos baseados em estimadores de distribuição e técnicas da otimização com muitos objetivos quando aplicados no contexto da refatoração automática de software.

Para que o objetivo do trabalho fosse alcançado, foi feita uma revisão bibliográfica sobre os EDA e refatoração de software baseada em busca. Após isso, foi implementada uma versão do EDA, capaz de resolver problemas de otimização combinatórios com muitos objetivos. Em linhas gerais, os EDA são algoritmos populacionais onde não são utilizados operadores de *crossover* e seleção, ao invés disso, modelos probabilísticos são utilizados na criação de uma nova população. Para que o EDA fosse capaz de lidar com problemas onde são considerados muitos objetivos, foram utilizados métodos de arquivamento na seleção das soluções que seriam utilizadas na geração do modelo probabilístico usado na criação da nova população. Uma nova população é criada a cada iteração do algoritmo e as iterações ocorrem até que um critério de parada é satisfeito. Ao fim da execução do algoritmo, é retornado um conjunto de soluções não dominadas.

Uma vez implementado o EDA, os experimentos realizados para avaliar o seu desempenho foram divididos em duas fases: a primeira fase consistiu na validação do algoritmo; na segunda fase, o algoritmo foi utilizado no contexto de SBSR. O algoritmo proposto foi validado no Problema da Mochila Multiobjetivo a fim de verificar se o EDA era capaz de resolver problemas de otimização com muitos objetivos. Os resultados obtidos na fase de validação do algoritmo foram publicados em (BOTELHO; BRITTO; SILVA, 2016).

Após a validação do algoritmo, o EDA foi utilizado para resolver o problema de busca por sequências de refatorações de software. Nesse contexto, métricas da Engenharia de Software são mapeadas como funções objetivo que são utilizadas para guiar a busca por boas soluções, o conjunto de métricas considerado neste trabalho foi o QMOOD. Para investigar o desempenho do EDA no contexto de SBSR, o algoritmo foi incorporado a um *framework* de refatoração

automática de software. Após a adição do EDA ao *framework* de refatoração, o algoritmo proposto foi testado em três softwares: Software Controle, Beaver e Gantt.

Nas duas fases dos experimentos, o indicador de qualidade utilizado na comparação dos algoritmos foi o hipervolume. Além disso, foram executados testes para verificar a existência de diferença estatística entre os resultados obtidos por cada um dos algoritmos, na primeira fase dos experimentos foi utilizado o teste de Friedman e na segunda fase foi utilizado o teste de Kruskal-Wallis.

Após a realização da primeira fase dos experimentos, percebeu-se que o EDA se destacava entre os demais algoritmos à medida que o número de objetivos do problema aumentava. O bom desempenho do algoritmo proposto no Problema da Mochila Multiobjetivo, deixou clara a capacidade do EDA para resolver problemas de otimização com muitos objetivos. Uma vez validado o algoritmo, a segunda fase dos experimentos foi realizada. Ao ser utilizado na busca por sequências de refatorações, o EDA foi superado pelo NSGA-II e pelo SPEA2. Após análise dos resultados, enxergamos como principal causa da queda de desempenho do EDA o aumento do número de valores possíveis para uma variável da solução do problema. Diante disso, com o modelo probabilístico utilizado, o algoritmo proposto não seria uma alternativa interessante para ser aplicada no contexto de SBSR em casos onde um conjunto grande de refatorações é considerado.

Tendo como base o trabalho desenvolvido até aqui, o objetivo agora é explorar outras técnicas utilizadas na otimização com muitos objetivos e combiná-las ao EDA. Além disso, pretende-se investigar novos modelos probabilísticos capazes de manter o bom desempenho do algoritmo à medida que o conjunto de valores possíveis para uma variável de decisão cresce. Outro ponto a ser explorado futuramente é a análise de desempenho do algoritmo considerando outra forma de comparação, que não seja o hipervolume. Por exemplo, comparar diretamente os valores obtidos por cada algoritmo para cada uma das funções objetivo.

Referências

ALETI, A. et al. Software architecture optimization methods: A systematic literature review. *IEEE Transactions On Software Engineering*, v. 39, n. 5, p. 658–683, May 2013. Citado na página 18.

BANSIYA, J.; DAVIS, C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 28, n. 1, p. 4–17, jan. 2002. Citado 3 vezes nas páginas 20, 23 e 24.

BENGOETXEA, E. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. Tese (Doutorado) — École Nationale Supérieure des Télécommunications, Paris, France, Dec 2002. Citado 2 vezes nas páginas 32 e 33.

BEUME, N. et al. On the complexity of computing the hypervolume indicator. *Evolutionary Computation, IEEE Transactions on*, v. 13, n. 5, p. 1075–1082, Oct 2009. ISSN 1089-778X. Citado na página 51.

BEZERRA, L. *Investigando a refatoração automática de software baseada em algoritmos de otimização multiobjetivos*. Dissertação (Mestrado) — Universidade Federal de Sergipe, Brazil, 2014. Citado 11 vezes nas páginas 14, 19, 21, 22, 23, 24, 25, 26, 27, 60 e 76.

BOTELHO, G.; BRITTO, A.; SILVA, L. A new estimation distributed algorithm applied to a many-objective discrete optimization problem. In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.: s.n.], 2016. p. 415–420. Citado 2 vezes nas páginas 46 e 65.

BOTELHO, G. et al. Investigating bioinspired strategies to solve large scale next release problem. In: *Proceedings of the XVIII Ibero American Conference on Software Engeneering*. [S.l.: s.n.], 2015. (CIBSE '15), p. 248–261. Citado na página 18.

BRITTO, A.; POZO, A. Using archiving methods to control convergence and diversity for many-objective problems in particle swarm optimization. In: *Evolutionary Computation (CEC), 2012 IEEE Congress on*. [S.l.: s.n.], 2012. p. 1–8. Citado na página 36.

BRITTO, A.; POZO, A. Using reference points to update the archive of MOPSO algorithms in many-objective optimization. *Neurocomput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 127, p. 78–87, mar. 2014. Citado na página 36.

CARVALHO, A. B. *Novas Estratégias para Otimização por Nuvem de Partículas Aplicadas a Problemas com Muitos Objetivos.* Tese (Doutorado) — Universidade Federal do Paraná, Curitiba, 2013. Citado 2 vezes nas páginas 31 e 50.

CHAVES-GONZÁLEZ, J. M.; PÉREZ-TOLEDANO, M. A. Differential evolution with Pareto tournament for the multi-objective next release problem. *Applied Mathematics and Computation*, v. 252, p. 1–13, February 2015. Citado na página 18.

CHIDAMBER, S. R.; KEMERER, C. F. Towards a metrics suite for object oriented design. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 26, n. 11, p. 197–211, nov. 1991. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/118014.117970>. Citado na página 20.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 0387332545. Citado 4 vezes nas páginas 28, 29, 30 e 31.

CUNNINGHAM, W. The wycash portfolio management system. In: *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*. New York, NY, USA: ACM, 1992. (OOPSLA '92), p. 29–30. Citado na página 18.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *Evolutionary Computation, IEEE Transactions on*, v. 18, n. 4, p. 577–601, Aug 2014. Citado 3 vezes nas páginas 34, 36 e 50.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Trans. Evol. Comp*, IEEE Press, Piscataway, NJ, USA, v. 6, n. 2, p. 182–197, abr. 2002. Citado na página 14.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Trans. Evol. Comp*, IEEE Press, Piscataway, NJ, USA, v. 6, n. 2, p. 182–197, abr. 2002. ISSN 1089-778X. Citado 2 vezes nas páginas 26 e 31.

DORIGO, M.; STÜTZLE, T. *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004. ISBN 0262042193. Citado na página 17.

DURILLO, J. J.; NEBRO, A. J. jmetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, v. 42, p. 760–771, 2011. Citado 4 vezes nas páginas 14, 26, 39 e 50.

EMMERICH, M.; BEUME, N.; NAUJOKS, B. An EMO algorithm using the hypervolume measure as selection criterion. In: *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*. Berlin, Heidelberg: Springer-Verlag, 2005. (EMO'05), p. 62–76. ISBN 3-540-24983-4, 978-3-540-24983-2. Citado na página 14.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2nd. ed. Boston, MA, USA: PWS Publishing Co., 1998. ISBN 0534954251. Citado na página 24.

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999. Citado 3 vezes nas páginas 18, 74 e 75.

GOUES, C. L. et al. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In: *Proceedings of the 34th International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2012. (ICSE '12), p. 3–13. ISBN 978-1-4673-1067-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2337223.2337225>. Citado na página 18.

HARMAN, M. Computational science – ICCS 2006: 6th international conference, reading, uk, may 28-31, 2006, proceedings, part iv. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. cap. Search Based Software Engineering, p. 740–747. Citado 2 vezes nas páginas 13 e 17.

ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Evolutionary many-objective optimization: A short review. In: *IEEE Congress on Evolutionary Computation*. [S.l.]: IEEE, 2008. p. 2419–2426. Citado 2 vezes nas páginas 14 e 35.

KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995. p. 1942–1948. Citado na página 14.

KENNEDY, J.; EBERHART, R. C. *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN 1-55860-595-9. Citado na página 32.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *SCIENCE*, v. 220, n. 4598, p. 671–680, 1983. Citado na página 17.

KIRLIK, G. *Test Instances for Multiobjective Discrete Optimization Problems*. 2014. <http://http://home.ku.edu.tr/~moolibrary/>. Acessado em 13/01/2016. Citado na página 52.

KIRLIK, G.; SAYıN, S. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, v. 232, n. 3, p. 479 – 488, 2014. Citado na página 52.

KÖPPEN, M.; YOSHIDA, K. Substitute distance assignments in NSGA-II for handling many-objective optimization problems. In: *Proceedings of the 4th International Conference on Evolutionary Multi-criterion Optimization*. Berlin, Heidelberg: Springer-Verlag, 2007. (EMO'07), p. 727–741. ISBN 978-3-540-70927-5. Citado na página 14.

KUMARI, A. C.; SRINIVAS, K. Article: Search-based software requirements selection: A case study. *International Journal of Computer Applications*, v. 64, n. 21, p. 28–34, 2013. Citado na página 18.

LANGDON, W. B.; HARMAN, M. Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 2014. Citado na página 18.

LORENZ, M.; KIDD, J. *Object-oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-179292-X. Citado na página 20.

MARIANI, T.; VERGILIO, S. R. A systematic review on search-based refactoring. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 83, n. C, p. 14–34, mar. 2017. ISSN 0950-5849. Disponível em: <https://doi.org/10.1016/j.infsof.2016.11.009>. Citado na página 13.

MICHURA, J.; CAPRETZ, M.; WANG, S. Extension of object-oriented metrics suite for software maintenance. *ISRN Software Engineering*, v. 2013, 2013. Citado na página 20.

MITCHELL, M. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262631857. Citado na página 17.

MKAOUER, M. W. et al. High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2014. (GECCO '14), p. 1263–1270. ISBN 978-1-4503-2662-9. Disponível em: <http://doi.acm.org/10.1145/2576768.2598366>. Citado na página 24.

MKAOUER, M. W. et al. Software refactoring under uncertainty: A robust multi-objective approach. In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2014. (GECCO Comp '14), p. 187–188. ISBN 978-1-4503-2881-4. Disponível em: <http://doi.acm.org/10.1145/2598394.2598499>. Citado na página 24.

MÜHLENBEIN, H. The equation for response to selection and its use for prediction. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 5, n. 3, p. 303–346, set. 1997. ISSN 1063-6560. Disponível em: <http://dx.doi.org/10.1162/evco.1997.5.3.303>. Citado 2 vezes nas páginas 34 e 42.

MUHLENBEIN, H.; MAHNIG, T. The factorized distribution algorithm for additively decomposed functions. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. [S.l.: s.n.], 1999. v. 1, p. 759 Vol. 1. Citado na página 34.

MURPHY-HILL, E.; PARNIN, C.; BLACK, A. How we refactor, and how we know it. In: *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. [S.l.: s.n.], 2009. p. 287–297. Citado na página 19.

NEBRO, A. J. et al. SMPSO: A New PSO-based Metaheuristic for Multi-objective Optimization. In: *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MCDM 2009)*. [S.l.]: IEEE Press, 2009. p. 66–73. Citado na página 32.

O'KEEFFE, M.; CINNÉIDE, M. O. Search-based software maintenance. In: *Proceedings of the Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2006. (CSMR '06), p. 249–260. ISBN 0-7695-2536-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1116163.1116409>. Citado 5 vezes nas páginas 14, 15, 18, 23 e 38.

O'KEEFFE, M.; CINNÉIDE, M. O. Search-based refactoring: An empirical study. *J. Softw. Maint. Evol.*, John Wiley & Sons, Inc., New York, NY, USA, v. 20, n. 5, p. 345–364, set. 2008. ISSN 1532-060X. Disponível em: <http://dx.doi.org/10.1002/smr.v20:5>. Citado 5 vezes nas páginas 14, 15, 18, 23 e 38.

O'KEEFFE, M.; CINNÉIDE, M. O. Search-based refactoring for software maintenance. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 81, n. 4, p. 502–516, abr. 2008. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2007.06.003>. Citado 4 vezes nas páginas 14, 15, 23 e 38.

O'KEEFFE, M. K.; CINNÉIDE, M. O. Getting the most from search-based refactoring. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2007. (GECCO '07), p. 1114–1120. ISBN 978-1-59593-697-4. Disponível em: <http://doi.acm.org/10.1145/1276958.1277177>. Citado 4 vezes nas páginas 14, 15, 18 e 23.

PARETO, V. *Cours d'Economie Politique*. Genève: Droz, 1896. Citado na página 29.

PAUL, T. K.; IBA, H. *Linear and Combinatorial Optimizations by Estimation of Distribution Algorithms*. 2002. Citado 4 vezes nas páginas 33, 42, 43 e 64.

PELIKAN, M.; GOLDBERG, D. E.; CANTÚ-PAZ, E. E. Linkage problem, distribution estimation, and bayesian networks. *Evol. Comput.*, MIT Press, Cambridge, MA, USA, v. 8, n. 3, p. 311–340, set. 2000. ISSN 1063-6560. Disponível em: <http://dx.doi.org/10.1162/106365600750078808>. Citado na página 34.

PELIKAN, M.; MUEHLENBEIN, H. The bivariate marginal distribution algorithm. In: ROY, R.; FURUHASHI, T.; CHAWDHRY, P. (Ed.). *Advances in Soft Computing*. Springer London, 1999. p. 521–535. ISBN 978-1-85233-062-0. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-0819-1_39>. Citado na página 34.

PELIKAN, M.; SASTRY, K.; GOLDBERG, D. E. Scalable optimization via probabilistic modeling. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. cap. Multiobjective Estimation of Distribution Algorithms, p. 223–248. ISBN 978-3-540-34954-9. Disponível em: <http://dx.doi.org/10.1007/978-3-540-34954-9_10>. Citado na página 34.

PRADITWONG, K. Solving software module clustering problem by evolutionary algorithms. In: *Proceedings of the 8th International Joint Conference on Computer Science and Software Engineering (JCSSE '11)*. Nakhon Pathom, Thailand: IEEE, 2011. p. 154–159. Citado na página 18.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2012. ISBN 3-900051-07-0. Disponível em: <http://www.R-project.org>. Citado na página 51.

SATO, H.; AGUIRRE, H. E.; TANAKA, K. Controlling dominance area of solutions and its impact on the performance of moeas. In: ____. Berlin: Springer, 2007. (Lecture Notes in Computer Science 4403: Evolutionary Multi-Criterion Optimization), p. 5–20. Citado na página 14.

SENG, O.; STAMMEL, J.; BURKHART, D. Search-based determination of refactorings for improving the class structure of object-oriented systems. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2006. (GECCO '06), p. 1909–1916. ISBN 1-59593-186-4. Disponível em: <http://doi.acm.org/10.1145/1143997.1144315>. Citado na página 23.

SILVA, L. de; BALASUBRAMANIAM, D. Controlling software architecture erosion: A survey. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 85, n. 1, p. 132–151, jan. 2012. ISSN 0164-1212. Citado na página 18.

STAUNTON, J.; CLARK, J. A. Applications of model reuse when using estimation of distribution algorithms to test concurrent software. In: ____. *Search Based Software Engineering: Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 97–111. ISBN 978-3-642-23716-4. Disponível em: <http://dx.doi.org/10.1007/978-3-642-23716-4_12>. Citado na página 14.

STAUNTON, J.; CLARK, J. A. Finding short counterexamples in promela models using estimation of distribution algorithms. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2011. (GECCO '11), p. 1923–1930. ISBN 978-1-4503-0557-0. Disponível em: <http://doi.acm.org/10.1145/2001576.2001834>. Citado na página 14.

TAKAHASHI, R. H. C. *Otimização Escalar e Vetorial - Vol. 1: Conceitos preliminares*. Belo Horizonte: [s.n.], 2007. Citado 2 vezes nas páginas 17 e 28.

TSUTSUI, S. Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram. In: . [S.l.: s.n.], 2002. p. 224–233. Citado 2 vezes nas páginas 34 e 43.

TZOREF, R.; UR, S.; YOM-TOV, E. Instrumenting where it hurts: An automatic concurrent debugging technique. In: *Proceedings of the 2007 International Symposium on Software Testing and Analysis*. London, United Kingdom: ACM, 2007. p. 27–38. Citado na página 18.

YOM-TOV, E. et al. Automatic debugging of concurrent programs through active sampling of low dimensional random projections. In: *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE '08)*. [S.l.]: IEEE, 2008. p. 307–316. Citado na página 18.

ZITZLER, E.; KÜNZLI, S. Indicator-based selection in multiobjective search. In: *in Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII*. [S.l.]: Springer, 2004. p. 832–842. Citado na página 14.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In: AL., K. G. et (Ed.). *EUROGEN 2001*. [S.l.: s.n.], 2002. p. 95–100. Citado na página 46.

Apêndices

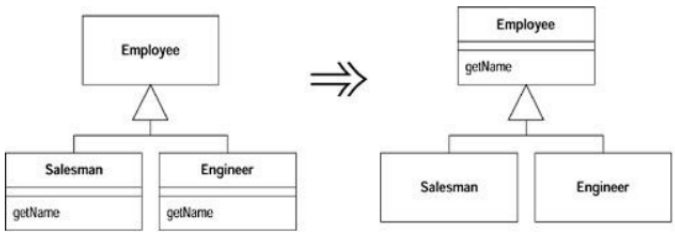
APÊNDICE A – Refatorações

Neste apêndice são apresentadas as descrições de todas as refatorações implementadas no *framework* na Seção 2.1.3. As descrições das refatorações foram originalmente apresentadas for Fowler (1999).

A.1 *Pull Up Method*

Objetiva reduzir a duplicação de código quando duas subclasses possuem um mesmo método, para isso, o método duplicado é movido para a superclasse. Na Figura 12, o método *getName* é movido das subclasses *Salesman* e *Engineer* para a superclasse *Employee*.

Figura 12 – Esquema geral da refatoração *Pull Up Method*.

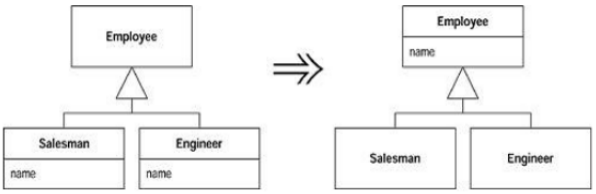


Fonte: Fowler (1999)

A.2 *Pull Up Field*

Objetiva reduzir a duplicação de código quando duas subclasses possuem um mesmo campo, para isso, o campo duplicado é movido para a superclasse. Na Figura 13, o campo *name* é movido das subclasses *Salesman* e *Engineer* para a superclasse *Employee*.

Figura 13 – Esquema geral da refatoração *Pull Up Field*.

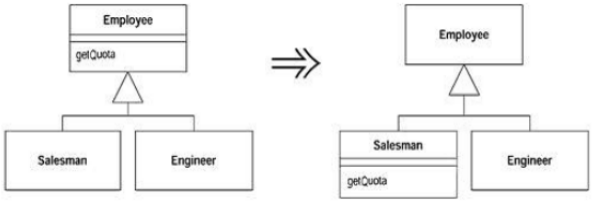


Fonte: Fowler (1999)

A.3 *Push Down Method*

Esta refatoração é feita quando um método está declarado em uma superclasse e é relevante apenas para algumas das subclasses. Após a aplicação da refatoração, o método é movido da superclasse para a subclasse. Na Figura 14, o método *getQuota* é movido da superclasse *Employee* para a subclasse *Salesman*.

Figura 14 – Esquema geral da refatoração *Push Down Method*.

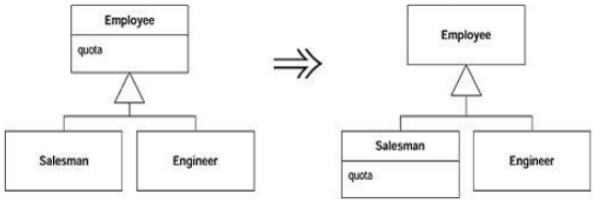


Fonte: Fowler (1999)

A.4 *Push Down Field*

Esta refatoração é feita quando um método está declarado em uma superclasse e não é relevante para todas as subclasses. Após a aplicação da refatoração, o método é movido da superclasse para a subclasse. Na Figura 15, o campo *quota* é movido da superclasse *Employee* para a subclasse *Salesman*.

Figura 15 – Esquema geral da refatoração *Push Down Field*.

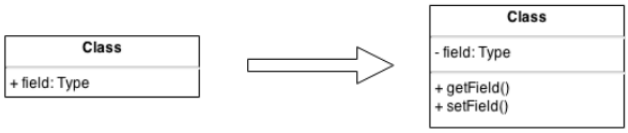


Fonte: Fowler (1999)

A.5 *Self Encapsulate Field*

Retira o acesso direto ao campo, após a aplicação desta refatoração, o acesso ao campo é feito de forma indireta, através de métodos acessores. Na Figura 16, o campo *field* deixa de ser acessado diretamente e passa a ser acessado através dos métodos *getField* e *setField*.

Figura 16 – Esquema geral da refatoração *Self Encapsulate Field*.



Fonte: Bezerra (2014)

A.6 *Increase Method Visibility*

Torna um método público. Esta refatoração destina-se a aumentar a interface externa da classe.