VINÍCIUS RENAN DE CARVALHO

UMA HIPER-HEURÍSTICA DE SELEÇÃO BASEADA EM DECOMPOSIÇÃO PARA ESTABELECER SEQUÊNCIAS DE MÓDULOS PARA O TESTE DE SOFTWARE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA - PR

2015

VINÍCIUS RENAN DE CARVALHO

UMA HIPER-HEURÍSTICA DE SELEÇÃO BASEADA EM DECOMPOSIÇÃO PARA ESTABELECER SEQUÊNCIAS DE MÓDULOS PARA O TESTE DE SOFTWARE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profa. Dra. Silvia Regina Vergilio

CURITIBA - PR

2015

C331h Carvalho, Vinícius Renan de
Uma hiper-heurística de seleção baseada em decomposição para estabelecer sequências de módulos para o teste de software/ Vinícius Renan de Carvalho. – Curitiba, 2015.
115 f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2015.

Orientador: Silvia Regina Vergilio .
Bibliografia: p. 82-88.

1. Algoritmos. 2. Engenharia de software. 3. Heurística. I. Universidade Federal do Paraná. II.Vergilio, Silvia Regina. III. Título.

CDD: 005.1

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Vinicius Renan de Carvalho, avaliamos o trabalho intitulado, "Uma Hiper-heurística de seleção baseada em decomposição para estabelecer sequências de módulos para o teste de software", cuja defesa foi realizada no dia 03 de dezembro de 2015, às 15:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela:

(X) **aprovação** do candidato. ( ) **reprovação** do candidato.

Curitiba, 03 de dezembro de 2015.

Profª. Dra. Silvia Regina Vergilio
PPGInf - Orientadora

Prof. Dr.Gledson Elias
UFPB – Membro Externo

Profª. Dra Aurora Trinidad Ramirez Pozo
PPGInf – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

À toda minha família, especialmente ao meu primo Carlos e sua esposa Raquel pelas inúmeras conversas durante nosso convívio, ao meu padastro Joel por sempre se mostrar preocupado em ajudar, e principalmente à minha mãe Natalina por desde pequeno me incentivar a estudar e por muitas vezes durante estes anos de estudo se preocupar e tentar me ajudar, mesmo não tendo conhecimentos na área de informática.

À Dayane pela compreensão e constante apoio durante o período do mestrado, me incentivando em momentos onde a motivação era pequena e sendo sempre companheira e presente em momentos mais tristes.

À meus companheiros laboratório, em especial Giovani, Olacir e Alexandre por inúmeras conversas sobre diversos assuntos e sobre assuntos relacionados ao trabalho, em muitas ocasiões respondendo duvidas.

À professora Drª Aurora por sempre que solicitada não sem importar em ensinar e ajudar na direção de minha pesquisa.

À minha orientadora Drª Silvia por todo conhecimento compartilhado e pela grande paciência demonstrada. Sinto que fui privilegiado em ter sido orientado por ela.

Aos amigos Marlon, Roberto e Vitor pelo grande companheirismo e lealdade demonstrados.

À CAPES, pelo apoio financeiro, indispensável para o desenvolvimento deste trabalho.

Aos Beatles, Helloween, Angra e Stratovarius que com suas musicas contribuíram para a conclusão deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Algoritmos multiobjetivos têm sido amplamente utilizados na busca de soluções de diversos problemas da computação, e mais especificamente para resolver problemas de Engenharia de Software na área conhecida como SBSE (*Search Based Software Engineering*). Contudo, conforme são intensificadas as aplicações destes algoritmos, tem-se a dificuldade de determinar qual algoritmo ou quais operadores são os mais indicados para um dado problema. Neste cenário as hiper-heurísticas são usadas para que o processo de busca seja guiado de forma que o melhor operador para o problema seja escolhido automaticamente. Neste contexto, destaca-se a hiper-heurística chamada HITO (*Hyper-heuristic for the Integration and Test Order Problem*), proposta para resolver o problema de estabelecer uma sequência de módulos para o teste de integração (ITO - *Integration and Test Order problem*). Em experimentos, a HITO obteve bons resultados, no entanto, existe a dificuldade para utilizar a HITO em conjunto com algoritmos baseados em decomposição, tais como o MOEA/D e MOEA/D-DRA. Estes algoritmos têm se mostrado bastante competitivos na literatura. Tendo este fato como motivação, este trabalho introduz uma hiper-heurística chamada HITO-DA (*Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach*) que propõe uma adaptação na HITO para permitir seu uso com algoritmos baseados em decomposição, na busca de soluções para o problema ITO. A HITO-DA foi instanciada com a meta-heurística MOEA/D-DRA usando o algoritmo de seleção FRRMAB (*Fitness Rate Rank Multi Armed Bandit*), e um novo algoritmo de seleção FRRCF (*Fitness Rate Rank with Choice Function*), proposto neste trabalho, que combina características do FRRMAB e CF (*Choice Function*). No estudo empírico conduzido a HITO-DA obteve melhores resultados do que a meta-heurística MOEA/D em todos os casos, e melhor desempenho em sistemas maiores, quando comparada com a HITO.

ABSTRACT

Multi-objective algorithms have been widely applied to find solutions in several problems, more specifically to solve Software Engineering problems, in the field called SBSE (Search Based Software Engineering). However, while these applications are intensified, we find some difficulty to select the most suitable operator for a problem. In this given scenario, hyper-heuristics are used to guide the search process in order to find the most suitable operator for a given problem. In this context, we find a hyper-heuristic, called HITO (*Hyper-heuristic for the Integration and Test Order problem*), proposed to solve the Integration and Test Order problem (ITO). HITO obtained good results, however, to adapt HITO to work with decomposition based algorithms, such as MOEA/D and MOEA/D-DRA, is a hard task. In the literature, these algorithms have shown competitive results. Based on this motivation, this work introduces a new hyper-heuristic called HITO-DA (Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach) that adapts HITO to work with decomposition based algorithms and to solve the ITO problem. The HITO-DA was instantiated using the algorithms MOEA/D-DRA, using the selection algorithm FRRMAB (Fitness Rate Rank Multi Armed Bandit) and a new algorithm, introduced in this work, named FRRCF (Fitness Rate Rank with Choice Function). FRRCF combines characteristics of the algorithms FRRMAB and CF (Choice Function). The conducted empirical study shows that HITO-DA obtained better results than MOEA/D in all cases, and obtained better results than HITO, in bigger systems.

CAPÍTULO 1

INTRODUÇÃO

1.1 Contexto

Estabelecer uma sequência de módulos para o teste de integração é um problema da Engenharia de Software, que visa a determinar a ordem na qual os módulos devem ser integrados e testados. Isto pode depender da disponibilidade prévia de um módulo, do qual outros módulos dependem, e caso este módulo não esteja disponível, faz-se necessária a construção de um *stub*.

Um *stub* é um pseudo-recurso que emula o comportamento de um recurso inexistente, sendo este criado quando um recurso é requerido por outro, mas este ainda não foi implementado. Assim um recurso r_1 pode ser desenvolvido sem o recurso r_2. A criação de um *stub* tem um custo envolvido, este custo deve ser minimizado. Isto pode ser feito decidindo qual *stub* deve ser implementado. Entretanto, esta tarefa não é trivial.

Diversas abordagens têm sido propostas para solucionar este problema, conhecido na literatura como problema ITO (*Integration Test Order problem*). Dentre estas, destaca-se a MOCAITO [7] (*Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem*) que trata este problema como um problema multiobjetivo, e por esta razão, trabalha com algoritmos multiobjetivos. Entretanto, para se empregar uma abordagem multiobjetivo algumas dificuldades existem, tais como: a escolha de operadores (operadores de mutação e cruzamento), e quais os parâmetros do algoritmo (como tamanho da população e probabilidades de mutação e cruzamento). Uma má escolha de operadores resulta em um impacto negativo, impedindo que melhores resultados sejam alcançados. Neste contexto, o conceito de hiper-heurística possui um papel interessante, pois retira do tomador de decisão parte do processo de escolha de configurações.

Uma hiper-heurística pode ser definida como [13]: (i) metodologias de seleção de heurísticas: (meta-)heurísticas para escolher (meta-)heurísticas, e (ii) metodologias de geração de heurísticas: (meta-)heurísticas para gerar novas (meta-)heurísticas.

Algoritmos multiobjetivos têm obtido bons resultados na área de Engenharia de Software baseada em busca (*Search Based Software Engineering* (SBSE)). O uso de hiper-heurísticas desperta interesse da comunidade de SBSE [30], mas ainda são poucos os trabalhos que tratam deste tópico.

Jia et al. [32] introduziram uma hiper-heurística para aprender e aplicar estratégias de teste combinatorial. O objetivo é obter uma algoritmo genérico para aplicar este tipo de teste.

Basgalupp et al. [9] propuseram uma hiper-heurística para a geração de algoritmos que criam árvores de decisão, a serem utilizadas na predição de esforço de software.

Kumari e Srinivas [37] propuseram uma hiper-heurística para trabalhar com o problema de agrupamento de módulos, a fim de selecionar heurísticas de baixo nível (*Low Level Heuristic* (LLH)) tais como operadores de mutação e cruzamento.

Para resolver o problema ITO, mencionado anteriormente foi proposta a hiper-heurística HITO (*Hyper-heuristic for the Integration and Test Order Problem*) [28, 29] para selecionar as LLHs que mais contribuam para a melhora da qualidade durante o processo evolutivo. Para isto, esta hiper-heurística aplicou os algoritmos de seleção UCB (*Upper Confidence Bound*) [24] e CF (*Choice Function*) [41] em conjunto com as meta-heurísticas NSGA-II (*Non Dominated Sorting Genetic Algorithm II*) [19] e SPEA2 (*Strength Pareto Evolutionary Algorithm*) [57].

As meta-heurísticas NSGA-II e SPEA2 usadas em conjunto com a hiper-heurística HITO são algoritmos que têm sido aplicados em diversos trabalhos da literatura [16]. No entanto a meta-heurística MOEA/D-DRA (*Multi-Objective Evolutionary Algorithm based on Decomposition with Dynamical Resource Allocation*) [54] tem se destacado como uma das melhores meta-heurísticas devido a seu desempenho no *CEC 2009 MOEA Contest* [55], e desta forma motiva o estudo de sua aplicação. Entretanto, a hiper-heurística

HITO requer adaptações para trabalhar com algoritmos multiobjetivos baseados em decomposição, dificultando assim sua aplicação em conjunto com o MOEA/D-DRA.

1.2 Objetivos

O uso de algoritmos baseados em decomposição pode contribuir para a melhora dos resultados existentes, desta forma uma hiper-heurística que permita a aplicação desta classe de algoritmos pode obter resultados ainda melhores. Devido ao fato de que a HITO não foi projetada tendo em mente algoritmos desta categoria, tem-se a dificuldade de implementar algoritmos como o MOEA/D em conjunto com a HITO. Dado este contexto, este trabalho tem como objetivo propor a hiper-heurística HITO-DA (*Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach*). Esta hiper-heurística incorpora algumas características da hiper-heurística HITO, e adiciona a capacidade de trabalhar com algoritmos multiobjetivos baseados em decomposição como o MOEA/D-DRA. Além disso a HITO-DA utiliza o algoritmo de seleção FRRMAB (*Fitness Rate Rank Multi Armed Bandit*) [40], uma versão melhorada do UCB, e o algoritmo de seleção proposto neste trabalho, chamado FRRCF (*Fitness Rate Rank Choice Function*), que incorpora algumas passos do FRRMAB, mas usa a função de seleção CF.

1.3 Organização do Trabalho

Este trabalho de dissertação esta organizado da seguinte maneira:

Capítulo 2 – Fundamentação Teórica Neste capítulo são apresentados os conceitos de algoritmos multiobjetivos. São apresentados os algoritmos NSGA-II, SPEA2, MOEA/D e MOEA/D-DRA. Ainda neste capítulo é apresentado o conceito de hiper-heurística, são mostrados os algoritmos de seleção UCB, FRRMAB e CF, e alguns trabalhos da área de SBSE que utilizam hiper-heurísticas;

Capítulo 3 – Estabelecendo Sequências de Módulos para o Teste de Integração Neste capítulo é descrito o problema de estabelecimento de sequência de módulos

para o contexto de software orientado a objeto e de software orientado a aspecto. Ainda neste capítulo são apresentadas a abordagem MOCAITO e a hiper-heurística HITO, utilizados com base neste trabalho;

Capítulo 4 – HITO-DA Neste capítulo é proposta a hiper-heurística HITO-DA e o algoritmo de seleção FRRCF é introduzido;

Capítulo 5 – Estudo Experimental Este capítulo descreve o estudo empírico conduzido e a avaliação dos resultados obtidos;

Capítulo 6 – Conclusões Apresenta a conclusão e também alguns trabalhos futuros;

Apêndice A – Este apêndice apresenta os resultados obtidos pelo teste de parâmetros executado neste trabalho.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar a fundamentação teórica necessária para o desenvolvimento e entendimento deste trabalho, introduzindo conceitos sobre otimização multiobjetivo e hiper-heurísticas. A Seção 2.1 apresenta o conteúdo sobre otimização multiobjetivo e descreve quatro dos principais algoritmos multiobjetivos. A Seção 2.2 apresenta os conceitos de hiper-heurística. A Seção 2.3 descreve alguns dos principais algoritmos de seleção de heurísticas de baixo nível. A Seção 2.4 apresenta aplicações de hiper-heurísticas a problemas da área de Engenharia de Software. Por fim a Seção 2.5 apresenta as considerações finais do capítulo.

2.1 Meta-heurísticas multiobjetivo

A área da computação bio-inspirada é uma subárea da computação que utiliza a natureza como fonte de inspiração na construção de seus algoritmos. Nesta área estão incluídos diversos algoritmos, dentre eles os algoritmos evolutivos, tais como Algoritmos Genéticos [27] e Programação Genética [36], algoritmos de inteligência coletiva como o de Otimização por Enxame de Partículas (*Particle Swarm Otimization* (PSO)) [34] e Otimização por Colônia de Formigas (*Ant Colony Otimization* (ACO)) [22].

Estes algoritmos têm sido utilizados na busca de soluções para problemas de alto grau de complexidade e possuem como característica empregar o uso de uma função objetivo, que tem como propósito representar de forma quantitativa o quanto uma solução é boa. Uma função objetivo é modelada de acordo com o problema, e é usada para calcular o valor objetivo de cada solução gerada durante o processo evolutivo. Uma função objetivo pode representar problemas de um único objetivo ou mono-objetivo (como por exemplo encontrar o mínimo de uma função matemática), e problemas multiobjetivos (como por

exemplo procurar uma solução para a compra de um automóvel levando em consideração o preço, consumo de combustível e velocidade máxima).

Em problemas mono-objetivos o valor objetivo de uma solução pode ser dado por um número (inteiro ou decimal) que é resultado da função objetivo empregada. Em casos de minimização soluções com valores objetivos menores são consideradas melhores, já em casos de maximização, soluções com valores objetivos maiores são consideradas melhores. Em problemas multiobjetivos existem vários valores objetivo para cada solução, e desta forma, a comparação entre soluções deve considerá-los simultaneamente.

Em muitos casos um problema multiobjetivo é resolvido realizando-se uma transformação para um objetivo único através de uma função de agregação, que utiliza pesos para cada objetivo, e prioriza assim um objetivo em favor dos outros. No entanto, o sucesso da abordagem de soma com pesos é dependente de uma escolha prévia dos pesos para os objetivos, que geralmente não é um processo simples, partindo de um ponto de vista do tomador de decisão [1].

Neste cenário, abordagens que permitam a otimização de vários objetivos ao mesmo tempo foram definidas, geralmente utilizando alguma metodologia de comparação, como o conceito de dominância de Pareto [16].

O conceito de dominância de Pareto [44] originou-se da área de Economia, e possui inúmeras aplicações a problemas do mundo real e em diversas áreas, envolvendo a teoria dos jogos e engenharia [35]. Neste conceito uma dada solução A no espaço de decisão de um problema multiobjetivo é superior a outra solução B, se e somente se, a solução A é tão boa quanto a solução B, considerando todos os objetivos, e estritamente melhor que a solução B considerando pelo menos um objetivo [1]. Desta forma pode-se dizer que a solução A domina a solução B ($A \prec B$), e que a solução B é uma solução dominada. A Figura 2.1 apresenta um gráfico para um problema multiobjetivo de minimização com três soluções A, B e C, onde $A \prec \{B, C\}$, pois possui menores valores para ambos objetivos (1 e 2), já a solução B domina a solução C ($B \prec C$), pois apesar das soluções terem o mesmo valor para o objetivo 2, a solução B possui menor valor para o objetivo 1.

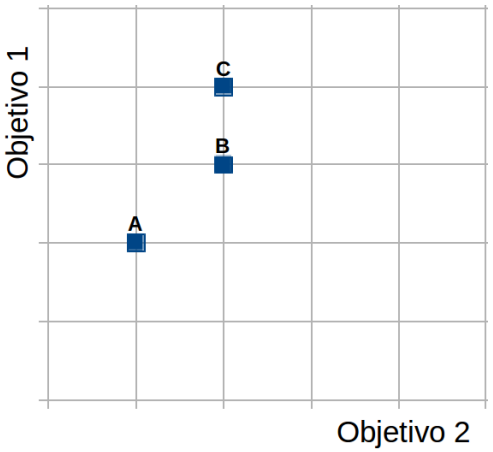


Figura 2.1: Dominância de Pareto em um problema com 2 objetivos

Define-se como fronteira de Pareto o conjunto de soluções não dominadas por nenhuma outra solução, para o qual não há relação de dominância entre os elementos deste conjunto. Assim, a fronteira de Pareto contém o conjunto de soluções para um dado problema, e diferentemente de uma abordagem mono-objetivo, onde tem-se apenas uma solução final, em uma abordagem multiobjetivo tem-se um conjunto de soluções, para que o tomador de decisões escolha a solução que mais atenda suas necessidades.

Para resolver problemas multiobjetivos, diversos algoritmos multiobjetivos podem ser empregados. Um mapeamento da área de SBSE (*Search Based Software Engineering*) [30] mostrou que os algoritmos evolutivos produzem bons resultados e são os mais utilizados. Isso se deve ao fato de que existem diferentes ferramentas que fornecem apoio a implementação, tais como o jMetal [23]. Por esta razão, este algoritmos são também utilizados neste trabalho. Dentre estes, destacam-se o NSGA-II (*Non-dominated Sorting Genetic Algorithm II*) [19] e o SPEA2 (*Strength Pareto Evolutionary Algorithm 2*) [57], aplicados em diversos trabalhos da área de SBSE, como por exemplo [7, 28, 29], e os algoritmos multiobjetivos MOEA/D (*Multi-Objective Evolutionary Algorithm based on Decomposition*) [53] e MOEA/D-DRA (*Multi-Objective Evolutionary Algorithm based on Decomposition with Dynamical Resource Allocation*) [54]. Estes algoritmos são apresentados a seguir.

2.1.1 Non Dominated Sorting Genetic Algorithm (NSGA-II)

O NSGA-II (*Non Dominated Sorting Genetic Algorithm II*) [19] é a implementação de um algoritmo genético que aplica o conceito de dominância de Pareto para problemas multiobjetivos. Este algoritmo cria uma população de indivíduos concorrentes, classifica e ordena cada indivíduo de acordo com seu nível de não dominância, aplica operadores evolutivos para criar um reservatório de indivíduos filhos, e então combina os pais e filhos antes de particionar a população em fronteiras [16].

O NSGA-II trabalha com uma população P_t (população P em um dado tempo t) e usa os operadores cruzamento e mutação para gerar novas soluções. Estas soluções compõem a população temporária Q_t (população Q em um dado tempo t), uma população que somente contém as soluções geradas. Ambas as populações possuem tamanho N, que é um parâmetro do algoritmo. Primeiramente, o algoritmo inicializa a população P_0 com soluções aleatórias e $Q_0 = \emptyset$, e em seguida gera novas soluções para compor a população temporária Q_1.

O algoritmo continua seu processo conforme a Figura 2.2, onde é realizada a união das populações ($R_t = T_t \cup Q_t$). Esta união possui tamanho igual a $2N$, e desta forma uma estratégia de substituições de soluções deve ser empregada para que o tamanho limite N seja respeitado.

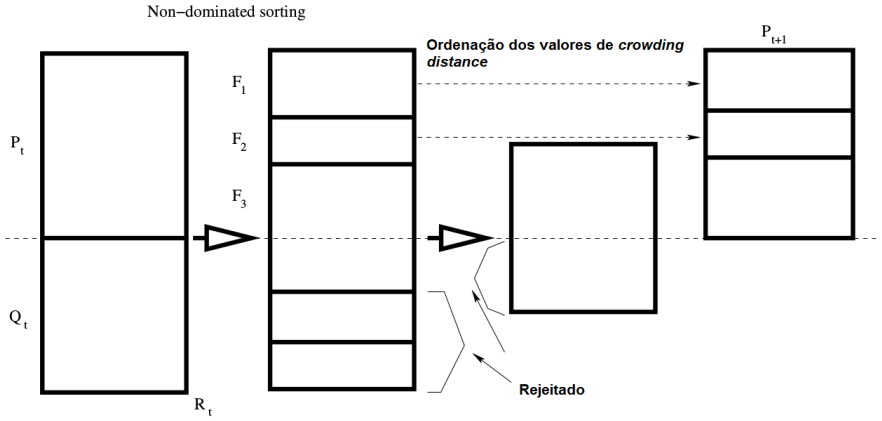


Figura 2.2: NSGA-II, adaptada de [16]

A estratégia de seleção de soluções usa o conjunto R_t para criar subconjuntos levando em consideração o valor de *rank* de uma solução. Este valor representa o quão dominada

é uma solução. Soluções que não são dominadas por nenhuma outra ($rank = 1$) são alocadas no subconjunto F_1, soluções dominadas por 1 solução ($rank = 2$) são alocadas no subconjunto F_2, e soluções dominadas por n soluções ($rank = n$) são alocadas no subconjunto F_{n+1}.

Posteriormente, o algoritmo adiciona as soluções contidas nos subconjuntos na nova população P_{t+1} até que sejam adicionadas N soluções na população P_{t+1}. Isto é feito primeiramente adicionando elementos do subconjunto F_1, depois elementos do conjunto F_2, e adicionando elementos dos subconjuntos seguintes até que a população P_{t+1} atinja o tamanho N, ou até que um subconjunto não possa mais ser completamente acomodado, pois violaria o tamanho máximo N (como o subconjunto F_3 na Figura 2.2). Assim, o NSGA-II usa o valor de *Crowding Distance* dos elementos deste subconjunto para selecionar quais elementos devem ser adicionados à população P_{t+1}. Soluções com maior valor de *Crowding Distance* são selecionadas e adicionadas na população P_{t+1} até que esta atinja o tamanho N.

O valor de *Crowding Distance* de uma determinada solução x é determinado pela distância Euclidiana da solução x de seus vizinhos. Esta métrica privilegia soluções mais espalhadas no espaço de busca, pois soluções mais dispersas possuem valores maiores de *Crowding distance*. Isto é feito pois soluções mais dispersas permitem maior exploração do espaço de busca.

O Algoritmo 1 apresenta o pseudocódigo do NSGA-II, que possui como entradas o tamanho da população N, as funções objetivo a serem otimizadas $f_k(x)$ e a quantidade máxima de gerações $maxGen$. Nas Linhas 6 e 12 são atribuídos os valores de $rank$ para as soluções que determinam o quão dominada a solução é. Na Linha 13 são gerados os subconjuntos baseados no valor de $rank$ das soluções, e para cada subconjunto gerado são calculados os valores de *Crowding Distance* para suas soluções (Linha 14), para que o subconjunto possa ser ordenado seguindo o critério de maior valor de *Crowding Distance* (Linha 15). Assim, na Linha 16 a nova população P_{t+1} é criada copiando subconjuntos inteiros até que não seja mais possível adicionar todas as soluções de um subconjunto, e caso a população P_{t+1} não contenha N soluções, então na Linha 18 os indivíduos do

último subconjunto (o que não foi possível adicionar inteiramente à nova população) são selecionados segundo seu valor de *Crowding Distance* e adicionados na nova população P_{t+1} até que esta atinja o tamanho N.

Algoritmo 1: Pseudocódigo do NSGA-II, adaptado de [16]

```
1  Entrada: N, maxGen, f_k(x)
2  início
3      Inicializa a população P';
4      Gerar população aleatória - Tamanho N';
5      Avaliar valores dos objetivos;
6      Atribuir rank baseado na dominância de Pareto - Ordenação;
7      Gerar população filho;
8          Seleção por torneio binário;
9          Cruzamento e Mutação;
10     para i ← 1 até maxGen faça
11         para Pai e Filho na Populacao faça
12             Atribuir rank baseado na dominância de Pareto - Ordenação;
13             Gerar subconjuntos de soluções não dominadas;
14             Atribuir valores de Crowding Distance;
15             Ordenar cada solução dos subconjuntos considerando o valor de Crowding Distance;
16             Percorrer todas os subconjuntos adicionando para a próxima geração do primeiro ao N'
               indivíduo;
17         fim
18         Selecionar melhores indivíduos do ultimo subconjuntos e com maior valor de Crowding Distance;
19         Criar nova geração;
20             Seleção por torneio binário;
21             Cruzamento e Mutação;
22     fim
23 fim
```

2.1.2 Strength Pareto Evolutionary Algorithm 2 (SPEA2)

O SPEA2 (*Strength Pareto Evolutionary Algorithm*) [57] possui como características o uso de uma população adicional denominada população ou arquivo externo, que possui como função armazenar as soluções não dominadas encontradas ao longo do processo evolutivo, e o cálculo de *fitness*, utilizado no processo de seleção de soluções empregadas na geração de novas soluções.

O valor de *fitness* de uma dada solução s é calculado segundo a Equação 2.1. Para isto, primeiramente são calculados os valores de *strength* para as soluções, de forma que o valor de *strength* de dada solução é obtido pela quantidade de soluções não dominadas que dominam esta solução. Posteriormente, é realizado para cada solução s, o somatório de valores de *strength* de todas as soluções dominadas por s, representado por $RawFitness$. No segundo elemento da Equação 2.1, é necessário realizar o cálculo da distância Eucli-

diana da solução s até o k-ésimo elemento da população, dado por $k = \sqrt{N + \overline{N}}$, onde N é o tamanho da população e $\overline{N}$ o tamanho do arquivo externo.

$$Fitness_s = RawFitness_s + \frac{1}{(\text{distância}(s,k) + 2)}, \tag{2.1}$$

O pseudocódigo do SPEA2 é apresentado no Algoritmo 2, que possui como parâmetros de entrada o tamanho da população N, o tamanho do arquivo externo $\overline{N}$, a quantidade máxima de gerações g, e a função a ser otimizada $f_k(x)$.

Algoritmo 2: Pseudocódigo do SPEA2, adaptado de [16]

```
1  Entrada: N, N̄, g, f_k(x)
2  início
3  |  Inicializa a população P';
4  |  Cria a população externa E';
5  |  para i ← 1 até g faça
6  |  |  Computar fitness de cada individuo em P' e E';
7  |  |  Copiar indivíduos com menor valor de fitness de P' e E' para E';
8  |  |  se Tamanho de E' maior que N̄ então
9  |  |  |  Usar operador de eliminação de soluções de E';
10 |  |  fim
11 |  |  senão se Tamanho de E' menor do que N̄ então
12 |  |  |  Usar soluções dominadas de P' para completar E';
13 |  |  fim
14 |  |  Executar seleção por torneio binário para preencher a mating pool;
15 |  |  Aplicar Cruzamento e Mutação para a mating pool;
16 |  fim
17 fim
```

O SPEA2 inicia seu processo computando o valor de *fitness* para todos os indivíduos na Linha 6, segundo a Equação 2.1. Em seguida as soluções com menor valor de *fitness* (não dominadas) são copiadas para o arquivo externo (Linha 4). Posteriormente é executada uma verificação para garantir que o arquivo possua exatamente $\overline{N}$ soluções, e caso o arquivo possua mais soluções do que o valor $\overline{N}$ (Linha 8) então um operador de eliminação de soluções é aplicado calculando-se a distância das soluções para seus vizinhos e removendo as soluções mais próximas. Caso o arquivo possua menos soluções do que o valor $\overline{N}$ (Linha 11) então o arquivo recebe soluções dominadas da população P'.

O SPEA2 continua seu processo selecionando pais de soluções em E' para compor o *mating pool* (Linha 14), que é um conjunto composto por soluções escolhidas via torneio binário, e usa este conjunto para gerar novas soluções através dos operadores de mutação e cruzamento (Linha 15).

2.1.3 Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D)

O MOEA/D (*Multi-Objective Evolutionary Algorithm based on Decomposition*) [53], é um algoritmo multiobjetivo baseado em decomposição. Este algoritmo possui como característica decompor um problema multiobjetivo em N subproblemas (processo de decomposição) para que estes subproblemas sejam trabalhados individualmente. Neste algoritmo cada um dos N subproblemas possui uma solução associada, que é a melhor solução encontrada para o dado subproblema.

No MOEA/D o valor de *fitness* de cada subproblema é dado por uma agregação ponderada dos m valores objetivos. Para isto, o algoritmo mantém um conjunto de vetores de peso e um conjunto de pontos ideais. O conjunto de vetores de peso é representado por $\Lambda = \{\lambda^1, ..., \lambda^N\}$, onde cada vetor λ é associado a um subproblema x e cada elemento do vetor λ é associado a um objetivo do subproblema x. Já o conjunto de pontos ideais é representado por $z^* = \{z_1^*, ..., z_m^*\}$, onde para cada um dos m objetivos, z_i^* é o melhor valor encontrado para um dado objetivo i.

Existem diversos métodos para realizar o processo de agregação, tais como a soma ponderada e o método *Tchebycheff* [10]. Na soma ponderada o problema é convertido em um problema de um objetivo por meio de uma combinação linear de seus objetivos. Já o método de agregação *Tchebycheff* (Equação 2.2) recebe como parâmetros um dado subproblema x, seu vetor de pesos λ e o conjunto de pontos ideais z^*. Nesta equação, para cada objetivo i do subproblema x, é realizado o cálculo da diferença absoluta entre o melhor valor obtido (z_i^* para o objetivo i) e o valor da função do objetivo i para o dado subproblema x ($f_i(x)$). O valor obtido pela diferença absoluta é multiplicado pelo peso correspondente ao objetivo i, ou seja, o valor é multiplicado pelo peso λ_i. Desta forma o maior valor obtido dentre os m resultados é então retornado como valor da Equação 2.2.

$$g^{te}(x|\lambda, z^*) = max\{\lambda_i|f_i(x) - z_i^*|\} \qquad 1 \leq i \leq m \tag{2.2}$$

onde x é o subproblema, λ o vetor de pesos para este subproblema, m é a quantidade de objetivos, z^* o conjunto de pontos ideais e $f_i(x)$ é o valor de *fitness* do objetivo i do subproblema x.

Para cada subproblema o MOEA/D mantém um conjunto B de subproblemas vizinhos, este conjunto é criado levando em consideração a distância Euclidiana entre os vetores de peso dos subproblemas. A quantidade de vizinhos de cada subproblema é determinada pelo parâmetro T presente no MOEA/D. O uso da vizinhança B é necessário para que cada subproblema seja otimizado levando em consideração informações de subproblemas vizinhos, pois subproblemas vizinhos que têm bom desempenho em um dado objetivo geralmente são bons candidatos em outros objetivos.

Além da vizinhança B, o MOEA/D eventualmente usa informações providas por subproblemas contidos na em toda população P (que contém todos subproblemas), onde a cada iteração o uso da vizinhança B ou população P é determinado aleatoriamente levando em consideração a probabilidade δ (parâmetro do algoritmo). Caso a probabilidade δ seja atingida, então a vizinhança B é usada, e caso contrário, então a população P é usada.

O MOEA/D gera uma nova solução y usando soluções associadas a subproblemas da vizinhança selecionada, e em seguida compara o valor de *fitness* da solução y gerada com os valores de *fitness* das soluções associadas aos subproblemas da vizinhança selecionada (ou população P). Posteriormente o algoritmo executa até Nr substituições de soluções caso o *fitness* de y seja melhor do que a solução corrente de um subproblema selecionado aleatoriamente da vizinhança selecionada (ou população P). Caso a solução y tenha o melhor valor ja encontrado em um dos m objetivos então o ponto z^* é atualizado.

O MOEA/D possui diversas variações, dentre elas destaca-se o MOEA/D-DRA por seu desempenho no *CEC 2009 MOEA Contest*, onde obteve o primeiro lugar [55] e devido a este desempenho, é usado neste trabalho. O MOEA/D-DRA é apresentado na próxima seção.

2.1.4 Multi-Objective Evolutionary Algorithm based on Decomposition with Dynamical Resource Allocation (MOEA/D-DRA)

No MOEA/D, todos os subproblemas são tratados igualmente e cada um deles recebe a mesma quantidade de esforço computacional. No entanto, estes subproblemas podem ter complexidades diferentes. Assim o MOEA/D-DRA (*Multi-Objective Evolutionary Algorithm based on Decomposition with Dynamical Resource Allocation*) [54] aloca diferentes recursos computacionais para os diferentes subproblemas de acordo com a utilidade destes. Esta utilidade leva em consideração a melhora obtida por um subproblema ao longo do processo evolutivo.

A utilidade π_x de um dado subproblema x é calculada segundo a Equação 2.3. Nesta equação o valor Δ^x é o decremento relativo, que é calculado levando em consideração os valores de *fitness* atual e anterior de um dado subproblema x.

$$\pi^x = \left\{ \begin{array}{l} 1 \text{ se } \Delta^x > 0.001; \\ (0.95 + 0.05\frac{\Delta^x}{0.001})\pi^x \text{ outro caso} \end{array} \right\} \quad (2.3)$$

$$onde$$

$$\Delta^x = \frac{VelhoFitness^x - NovoFitness^x}{VelhoFitness^x}$$

O Algoritmo 3 apresenta o pseudocódigo do MOEA/D-DRA, que recebe como parâmetros o tamanho da vizinhança T, a quantidade máxima de substituições Nr, o valor de δ, o valor de N (quantidade de subproblemas), e a quantidade máxima de gerações $maxGen$.

O algoritmo inicia seu processo com a inicialização da vizinhança principal P e soluções associadas, e os pontos ideais na Linha 3 e com a inicialização da vizinhança B para cada solução na Linha 6, onde cada vizinhança é criada com tamanho T levando em consideração os vetores de peso. Na Linha 7 a utilidade também é inicializada.

Enquanto o critério de parada $gen < maxGen$ não é atingido, o conjunto I é criado levando em consideração a utilidade dos subproblemas, e para cada solução em I uma nova solução y é gerada através da aplicação do operador evolutivo (cruzamento ou evolução diferencial) usando os pais $p1$ e $p2$ (Linha 20), onde $p1$ é a solução associada ao subproblema selecionado aleatoriamente em SP (P ou B) e $p2$ é solução do atual subproblema x. Posteriormente na Linha 21 o operador de mutação é aplicado em y.

Na Linha 22 o ponto de referência z^* é atualizado usando o novo valor y caso este possua algum objetivo com melhor valor que o mesmo objetivo no conjunto dos pontos ideais z^*. Na Linha 24 inicia-se uma repetição com o objetivo de realizar Nr substituições caso o valor de n na Linha 26 seja maior que 0, ou seja, caso y seja melhor que uma solução x^j selecionada aleatoriamente da vizinhança SP. O cálculo de n é realizado levando em consideração o valor da equação $Tchebycheff$ para o item gerado y e para o item selecionado aleatoriamente x^j, onde $n > 0$ significa que a solução y obteve melhoras em relação à solução x^j.

Posteriormente, o cálculo $mod(gen, 50)$ é realizado. Isto é feito para que a cada 50 gerações todos os valores de utilidade sejam recalculados (Linha 35).

Algoritmo 3: Pseudocódigo do MOEA/D-DRA

Entrada: T, Nr, δ, $maxGen$, N
início
    Inicializa a vizinhança P e os pontos ideais z^*;
    $gen = 0$;
    para $x \leftarrow 1$ ***até*** N **faça**
        $B(x) = \{i_1, ..., i_T\}$ onde $\lambda^{i_1}, ..., \lambda^{i_T}$ são T vetores mais próximos de λ^x ;
        $\pi^x = 1$;
    fim
    enquanto $gen < maxGen$ **faça**
        Formar o conjunto I levando em consideração o valor de utilidade π de cada subproblema;
        para cada $x \in I$ **faça**
            se $Random(0, 1) < \delta$ **então**
                SP=B(x);
            fim
            senão
                SP=P;
            fim
            $p1 \leftarrow$ Selecionar aleatoriamente item de SP;
            $p2 \leftarrow x$;
            $y \leftarrow op.apply(p1, p2)$;
            $mt.apply(y)$;
            Atualizar ponto de referência z^*;
            c=0;
            enquanto $c < Nr \| SP! = \emptyset$ **faça**
                aleatoriamente selecione uma solução x^j de SP;
                $n = \frac{g(x^j|\lambda^j, z^*) - g(y|\lambda^j, z^*)}{g(x^j|\lambda^j, z^*)}$;
                se $n > 0$ **então**
                    Substituir x^j por y;
                    Remover x^j de P;
                fim
                $c++$;
            fim
            $gen++$;
            se *mod(gen,50)==0* **então**
                atualizar a utilidade π^x para cada subproblema;
            fim
        fim
    fim
fim

2.2 Hiper-heurísticas

Meta-heurísticas têm sido amplamente utilizadas em busca de soluções de diversos problemas, contudo para aplicar uma meta-heurística a um problema é necessário conhecimento sobre o mesmo. Conforme vão surgindo aplicações de meta-heurísticas a novos problemas, tem-se também a dificuldade de determinar qual combinação de meta-heurísticas e operadores seriam os mais efetivos para o problema. Neste contexto, o conceito de hiper-heurística surge como uma possível solução para que um processo de busca seja guiado de forma a identificar heurísticas de baixo nível promissoras (*Low Level Heuristic* LLH) [50], onde uma LLH pode ser uma combinação de operadores (como por exemplo operadores de mutação e cruzamento), ou uma meta-heurística (como NSGA-II, SPEA2, etc.).

Segundo Cowling et al. [18] hiper-heurísticas são heurísticas que escolhem heurísticas. Posteriormente devido à expansão da área, uma nova definição foi proposta por Burke et al. [13], hiper-heurísticas portanto são: (i) metodologias de seleção de heurísticas: (meta-) heurísticas para escolher (meta-)heurísticas, e (ii) metodologias de geração de heurísticas: (meta-)heurísticas para gerar novas (meta-)heurísticas de dados componentes.

Neste trabalho o foco está em hiper-heurísticas de seleção. Assim, a escolha de uma LLH é realizada comparando os resultados obtidos com resultados anteriores, baseando-se em algum indicador de qualidade.

A seguir são apresentados o conceito de barreira de domínio apresentado, a classificação de hiper-heurísticas e algumas das principais hiper-heurísticas.

2.2.1 Barreira do domínio em hiper-heurísticas

O conceito de barreira de domínio, mostrado na Figura 2.3, tem como objetivo garantir que a hiper-heurística possua conhecimento somente de informações não ligadas ao domínio do problema. Desta forma a hiper-heurística tem acesso apenas a informações como a quantidade de LLHs e a melhora obtida, sendo esta quantificada por alguma métrica de qualidade.

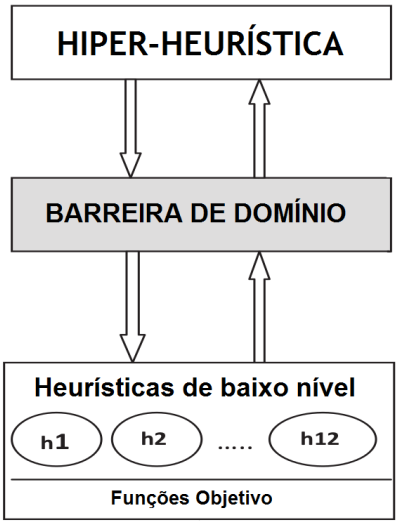


Figura 2.3: Barreira de domínio, adaptada de [13]

Este conceito busca garantir que uma hiper-heurística possa ser aplicada a novos problemas, bastando apenas a substituição da camada de domínio do problema implementado pelas heurísticas de baixo nível.

2.2.2 Classificação de Hiper-heurísticas

Pode-se classificar uma hiper-heurística segundo à sua natureza e sua capacidade de aprendizado como apresentado na Figura 2.4. Com relação a natureza pode-se classificar uma hiper-heurística como: metodologias de seleção, onde heurísticas existentes podem ser escolhidas, ou hiper-heurística de geração onde novas heurísticas são geradas através de componentes preexistentes [13].

Uma hiper-heurística pode também ser classificada como de natureza construtiva ou perturbativa. Heurísticas perturbativas trabalham considerando soluções candidatas completas e as altera através da modificação de um ou mais dos elementos destas soluções, já os métodos construtivos trabalham considerando parcialmente soluções candidatas, em que um ou mais elementos não estão disponíveis, e de forma iterativa, cria novos elementos através de elementos anteriores [13].

Com relação à capacidade de aprendizado de uma hiper-heurística existem as categorias: aprendizado online, onde o aprendizado é realizado enquanto o algoritmo esta

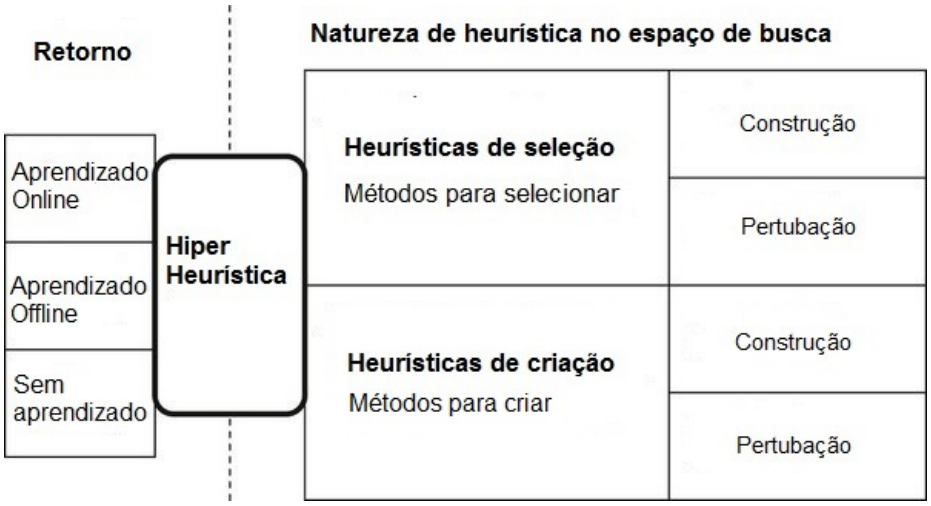


Figura 2.4: Tipos de hiper-heurísticas, adaptada de [13]

resolvendo a instancia de um problema; aprendizado off-line, onde é necessário que um conjunto de treinamento seja aplicado à hiper-heurística. O resultado deste treinamento é um algoritmo que potencialmente é mais apto em qualquer instância do problema para o qual foi treinado, e por fim o não aprendizado, como por exemplo, a escolha aleatória de heurísticas.

Neste trabalho o foco é dado em aprendizado online e nos seguintes algoritmos de seleção *Choice Function*, *Multi Armed Bandit* e *Fitness Rate Rank Multi Armed Bandit*, que são descritos a seguir.

2.3 Algoritmos de seleção de heurísticas de baixo nível

Existem diversos algoritmos geralmente utilizados com hiper-heurísticas, dentre eles destacam-se os algoritmos de seleção CF (*Choice Function*) [18] por sua simplicidade de implementação e melhora nos resultados obtidos e o FRRMAB (*Fitness Rate Rank Multi Armed Bandit*) [40] devido a seus recentes e promissores resultados na aplicação de hiper-heurísticas.

2.3.1 Choice Function (CF)

O algoritmo de seleção CF (*Choice Function*) teve sua aplicação inicialmente estudada por Cowling et al. [18] e adaptativamente classifica cada heurística de baixo nível com relação a

uma pontuação combinada [13]. Esta pontuação combinada é calculada segundo o valor de uma função matemática (Equação 2.4), onde o elemento f_1 representa o quão bom foram os resultados obtidos pela aplicação de uma dada LLH h_i, o elemento f_2 representa se a sequencia das LLHs h_i e h_j (executada anteriormente) é boa, e o elemento f_3 representa o tempo de espera (em segundos) que a LLH h_i foi mantida inativa. Este elemento tem como função uma melhor diversificação (exploração) na escolha de heurísticas de baixo nível. Caso este elemento seja desconsiderado, a equação da CF escolheria uma LLH no inicio do processo evolutivo e a manteria até o fim do processo evolutivo.

$$F(h_i) = \alpha f_1(h_i) + \beta f_2(h_j, h_i) + \delta f_3(h_i) \tag{2.4}$$

onde α, β e δ são parâmetros que regulam a equação.

Usar um alto valor para δ aumenta a influência do elemento f_3 e permite que as escolhas sejam feitas de forma mais uniforme, contudo impede que o desempenho de cada heurística seja relevante nas escolhas. Por outro lado, usar altos valores de α e β faz com que os elementos f_1 e f_2 tenham maior influência e faz com que as escolhas sejam mais elitistas e priorizem mais uma heurística em detrimento de outras.

Maashi et al. [41] propuseram uma simplificação para a CF, removendo o elemento $\beta f_2(h_j, h_i)$, e o parâmetro δ de $\delta f_3(h_i)$. Desta forma tem-se a nova CF descrita pela Equação 2.5.

$$F(h_i) = \alpha f_1(h_i) + f_2(h_i) \tag{2.5}$$

Na CF apresentada na Equação 2.5 o elemento f_1 possui a mesma função do elemento f_1 da Equação 2.4, e o elemento f_2 possui a mesma função do elemento f_3 da Equação 2.4. Esta CF será usada neste trabalho de mestrado, pois foi utilizada nos trabalho relacionados [28, 29].

2.3.2 Upper Confidence Bound (UCB)

O UCB (*Upper Confidence Bound*) proposto por Auer et al. [8], para solução do problema MAB (*Multi Armed Bandit*) estudado na teoria de jogos. Este problema estuda o dilema

entre a exploração (descoberta) e a explotação (intensificação). Neste problema um jogador tem como objetivo maximizar sua recompensa em um jogo com um conjunto de caça-níqueis. Para isto o jogador deve definir a ordem de execução e quantas vezes deve jogar em cada um dos caça-níqueis. Cada caça-níquel possui uma alavanca ou um braço (*arm*) que é acionada para executar o jogo. Assim, no problema MAB a escolha de um "caça-níquel" é representada pela escolha de um braço.

O UCB atribui uma heurística de baixo nível ou LLH (*Low Level Heuristic*) para cada um dos K braços com uma probabilidade desconhecida de obter uma recompensa q_i, onde i é o i-ésimo braço. Formalmente, uma dada LLH i é associada a: 1) uma recompensa empírica q_i (como por exemplo, a média das melhoras obtidas); e 2) um intervalo de confiança Nt_i, representado pelo número de vezes que um i-ésima LLH foi tentada.

O algoritmo UCB inicia realizando o processo de escolha de LLH, esta LLH é então executada. Em seguida o processo de atribuição de crédito é executado para atualizar os valores de q_i e Nt_i, que são valores usados no processo de escolha da heurística. No processo de escolha, a LLH que maximiza a Equação 2.6 é escolhida.

$$op = argmax_{i=1...K}(q_i + C * \sqrt{\frac{2 * \ln \sum_{j=1}^{K} Nt_j}{Nt_i}}) \tag{2.6}$$

onde C é o fator escalar especificado como parâmetro do algoritmo. Este fator possui como função ponderar se devem ser priorizadas LLHs com melhor recompensa ou que não tenham sido aplicadas constantemente, a variável K é a quantidade de LLHs existentes, e cada Nt_j representa a quantidade de vezes que uma dada LLH foi aplicada em um dado tempo.

O processo de atribuição de crédito (*Credit Assignment*) tem como função atribuir uma recompensa a uma LLH utilizando seu desempenho recente. Este desempenho leva em consideração a melhora obtida após aplicar esta LLH. A partir disto realiza o cálculo da recompensa q de acordo com uma das seguintes metodologias:

- *Last improvement*: utilizar a última melhora obtida e atribuir esta melhora como a recompensa;

- *Average reward*: utilizar a média de todas as melhoras obtidas e atribuir esta média como a recompensa;
- *Extreme reward*: utiliza o maior valor de melhora encontrada e atribuir este valor como a recompensa;
- *Normalized Extreme reward*: trabalha como o *Extreme reward*, mas utilizando normalização;

2.3.3 Fitness Rate Rank Multi Armed Bandit (FRRMAB)

O FRRMAB (*Fitness Rate Rank Multi Armed Bandit*) [40] é um algoritmo de seleção que incorpora alguns atributos do UCB e também possui duas etapas, sendo elas a escolha de LLHs e o método de atribuição de crédito (*Credit Assignment*). A escolha de LLHs é similar ao UCB, com a diferença que o UCB usa o valor obtido por uma das quatro metodologias descritas na Seção 2.3.2 como recompensa q na Equação 2.6, já o FRRMAB usa valores de FRR (*Fitness Rate Ranking*) como recompensa q.

Os valores de FRR são gerados pelo método de atribuição de crédito empregado pelo FRRMAB. Estes valores têm como função avaliar a qualidade de uma LLH de forma normalizada. Além disso, os valores de FRR são calculados de forma que a influência da LLH que tenha os melhores resultados possa ser incrementada.

Ambas as etapas de seleção de LLH e atribuição de crédito são descritas a seguir.

2.3.3.1 Atribuição de Crédito

A etapa de atribuição de crédito inicia pelo cálculo da melhora obtida para cada LLH realizado através da Equação 2.7. Neste algoritmo esta melhora recebe o nome de FIR (*Fitness Improvement Rate*). Este valor é o somatório das melhoras proporcionadas pelos filhos em relação aos pais ao aplicar uma dada LLH *op*. Nesta Equação $F(p)$ é o valor de *fitness* do pai p e $F(c)$ é o valor de *fitness* do filho c. A Equação 2.7 foi modelada para considerar um problema de minimização, pois recompensa o FIR do operador quando o filho c possui *fitness* menor do que o pai p.

$$n = \frac{(F(p) - F(c))}{F(p)}$$
$$FIR_{op} = FIR_{op} + n, n > 0 \tag{2.7}$$

O método de atribuição de crédito do FRRMAB incorpora o uso do conceito de janela de tempo empregado em [24]. Neste conceito, ao longo do processo evolutivo, os valores de FIR são armazenados em uma janela de tempo (um vetor do tipo fila) de forma que cada item desta janela possua um valor FIR e um índice que represente uma determinada LLH *op*. A janela de tempo possui um tamanho determinado pelo parâmetro W, e obedece a metodologia FIFO (*First In First Out*), ou seja, os itens sempre são adicionados ao final da lista, caso um novo item deva ser adicionado e a quantidade de itens na janela for igual a W, então o elemento mais antigo da lista deve ser removido para que o novo item seja adicionado no fim da lista. É valido ressaltar que uma dada LLH *op* pode possuir vários valores de FIR na janela de tempo, obtidos em diferentes momentos.

Posteriormente é obtido o *Reward* para cada LLH. Isto é realizado através do somatório dos valores de FIR (contidos na janela de tempo) relacionados a uma dada LLH. Em seguida o valor de *Rank* é obtido para cada LLH. Este valor é dado pelo ranqueamento decrescente do vetor *Reward* (vetor que contém o valor de *Reward* para todas as LLHs), onde a LLH que possui o maior valor no vetor *Reward* recebe $Rank = 1$ e a LLH com o menor valor recebe $Rank = K$.

Com os valores de *Reward* e *Rank* é possível obter o valor de *Decay* para cada LLH através da Equação 2.8. O valor de *Decay* é necessário para o cálculo do FRR, usado pelo método de seleção. O valor de $D \in [0, 1]$ é um parâmetro que regula a Equação 2.8, onde valores menores de D aumentam a influência da melhor LLH, resultando em escolhas mais elitistas.

$$Decay_{op} = D^{Rank_{op}} * Reward_{op} \tag{2.8}$$

Em seguida o valor de $DecaySum$ é calculado através da Equação 2.9, este valor é o somatório dos valores de $Decay$ de todas LLHs.

$$DecaySum = \sum_{op=1}^{K} Decay_{op} \tag{2.9}$$

A partir disto, o cálculo do valor de FRR é executado de acordo com a Equação 2.10.

$$FRR_{op} = \frac{Decay_{op}}{DecaySum} \tag{2.10}$$

O Algoritmo 4 mostra o funcionamento da etapa de atribuição de crédito para o FRRMAB. Este algoritmo recebe como parâmetros a janela de tempo ($SlidingWindow$), a quantidade de LLHs (K), e o parâmetro D que regula a Equação 2.8, usada no algoritmo. Este algoritmo tem como objetivo gerar os vetores FRR e Nt (quantidade de vezes que uma dada LLH foi executada na janela de tempo) para cada LLH. Isto é realizado usando os valores de FIR contidos na janela de tempo $SlidingWindow$.

O algoritmo inicia gerando o vetor auxiliar $Reward$, que é um somatório de todos os valores de FIR existentes na janela de tempo para uma dada LLH, e gerando o vetor Nt (Linha 10). Em seguida, os valores do vetor auxiliar $Reward$ são classificados em ordem decrescente (Linha 12) para que cada valor do vetor tenha uma posição (1 a K) na classificação $Rank$. Com os vetores $Reward$ e $Rank$, os valores $Decay$ são calculados (Linha 17 pela Equação 2.8). Posteriormente, o algoritmo executa o somatório dos valores contidos no vetor $Decay$ pela Equação 2.9 para que este somatório seja usado em conjunto com os valores contidos no vetor $Decay$ no cálculo do FRR. Este cálculo é executado para cada LLH segundo a Equação 2.10 (Linha 21). Ao final do algoritmo os vetores FRR e Nt são retornados.

2.3.3.2 Escolha de heurísticas de baixo nível

O Algoritmo 5 mostra o funcionamento da etapa de escolha de LLHs. Este algoritmo recebe como parâmetros o vetor FRR (gerado pela etapa de atribuição de crédito), K que representa a quantidade de LLHs, o vetor Nt (número de aplicações de uma LLH

Algoritmo 4: Pseudocódigo do método de atribuição de crédito (*Credit Assignment*) do FRRMAB

```
1  Entrada: SlidingWindow, D, K
2  Saída: FRR, Nt
3  início
4      Atribuir 0 para cada elemento do vetor Reward;
5      Atribuir 0 para cada elemento do vetor Nt;
6      para Item in SlidingWindow faça
7          FIR=Item.getFIR();
8          op=Item.getIndexOp();
9          Reward_op=Reward_op + FIR;
10         Nt_op++;
11     fim
12     Classificar vetor Reward em ordem decrescente;
13     para op ← 1 até K faça
14         Rank_op = Reward.getOpPosition(op);
15     fim
16     para op ← 1 até K faça
17         Decay_op = D^Rank_op * Reward_op;
18     fim
19     DecaySum = Σ_{op=1}^{K} Decay_op;
20     para op ← 1 até K faça
21         FRR_op = Decay_op/DecaySum;
22     fim
23     retorna FRR;
24 fim
```

na janela de tempo), e o parâmetro C usado na Equação 2.6 (Linha 8). Este algoritmo garante a execução de todas as LLHs por uma rodada, antes de escolher uma LLH pela Equação 2.6, assim garantindo uma escolha justa.

Algoritmo 5: Pseudocódigo do método de seleção do FRRMAB, adaptado de [40]

```
1  Entrada: FRR, C, K, Nt
2  Saída: op
3  início
4      se Existem LLHs ainda não selecionadas então
5          op=Selecionar aleatoriamente alguma LLH ainda não selecionada;
6      fim
7      senão
8          op = argmax_{i=1...K}(FRR_i + C * sqrt((2*ln Σ_{j=1}^{K} Nt_j) / Nt_i));
9      fim
10     retorna op;
11 fim
```

2.3.4 MOEA/D-FRRMAB

O MOEA/D-FRRMAB proposto em [40] aplica o algoritmo de seleção FRRMAB na escolha de LLHs para a meta-heurística MOEA/D-DRA. Assim todas as tarefas do FRRMAB são executadas conforme mostrado na Seção 2.3.3. Para isto, o valor de *Fitness* para cada

LLH é calculado segundo a Equação 2.11.

$$F = g(x^j|\lambda^j, z^*) \tag{2.11}$$

onde g é a função *Tchebycheff* $g(subproblema|\text{peso associado}, \text{pontos ideais})$

O Algoritmo 6 mostra o funcionamento do MOEA/D-FRRMAB. Este algoritmo é uma adaptação do MOEA/D-DRA [54] e desta forma incorpora todas as tarefas executadas pelo mesmo, com a diferença que o MOEA/D-DRA usa operadores de mutação e cruzamento previamente selecionados pelo usuário. Já o MOEA/D-FRRMAB usa um conjunto de LLHs (operador de cruzamento e operador de mutação), e devido a este fato, necessita de uma forma para selecionar qual LLH deve ser aplicada. Outras diferenças entre o MOEA/D-DRA e o MOEA/D-FRRMAB são o uso dos parâmetros C e D, usados pelo MAB, a variável W que define o tamanho máximo da janela, e a variável K que representa a quantidade de LLHs disponíveis.

No MOEA/D-FRRMAB a escolha da LLH a ser aplicada é realizada pelo método de seleção apresentado no Algoritmo 5 (Linha 22). Nas Linhas 30 e 34 é realizado o cálculo do FIR segundo a Equação 2.7 para que este valor seja adicionado a janela de tempo na Linha 39. Posteriormente o cálculo do FRR é executado na Linha 40, este cálculo é executado pelo Algoritmo 4.

2.4 Aplicações de hiper-heurísticas em Engenharia de Software

A área de Engenharia de Software Baseada em Busca (*Search-Based Software Engineering* (SBSE)) investiga a aplicação de técnicas baseadas em busca na solução de problemas complexos da área de Engenharia de Software. Na SBSE os problemas da Engenharia de Software são modelados como problemas de otimização. Nestes problemas o objetivo é minimizar ou maximizar uma função ou grupo de fatores [17]. Técnicas de otimização baseadas em busca geralmente são utilizadas, às quais estão associados dois aspectos: um espaço de busca, que contém todas as possíveis soluções para o problema; e uma função de *fitness* que avalia a qualidade da solução [17].

Algoritmo 6: Pseudocódigo do MOEA/D-FRRMAB

Entrada: T, Nr, δ, C, D, W, K, $maxGen$, N
início
  Inicializa a população P e os pontos ideais z^*;
  FRR $\leftarrow \emptyset$;
  Nt $\leftarrow \emptyset$;
  FIR $\leftarrow$ Inicializar vetor com valor 0 para cada *op*;
  SlidingWindow $\leftarrow$ Criar SlidingWindow com tamanho W;
  $gen = 0$;
  para $i \leftarrow 1$ ***até*** N **faça**
    $B(i) = \{i_1, ..., i_T\}$ onde $\lambda^{i_1}, ..., \lambda^{i_T}$ são T vetores mais próximos de λ^i ;
    $\pi^i = 1$;
  fim
  enquanto $gen < maxGen$ **faça**
    Formar o conjunto I levando em consideração o valor de utilidade π de cada subproblema;
    para cada $i \in I$ **faça**
      se $Random(0, 1) < \delta$ **então**
        SP=B(i);
      fim
      senão
        SP=P;
      fim
      op=MABSelector(FRR, C, K, Nt);
      $p1 \leftarrow$ Selecionar aleatoriamente item de SP;
      $p2 \leftarrow i$;
      $y \leftarrow op.apply(p1, p2)$;
      Atualizar ponto de referência z^*;
      c=0;
      enquanto $c < Nr \| P! = \emptyset$ **faça**
        aleatoriamente selecione uma solução x^j de SP;
        $n = \frac{g(x^j|\lambda^j, z^*) - g(y|\lambda^j, z^*)}{g(x^j|\lambda^j, z^*)}$;
        se $n > 0$ **então**
          Substituir x^j por y;
          Remover x^j de P;
          $FIR_{op} = FIR_{op} + n$;
        fim
        $c++$;
      fim
      SlidingWindow.setIndex(op);
      SlidingWindow.addFIR(FIR_{op});
      FRR=CreditAssignment(SlidingWindow, D, K, Nt);
      $gen++$;
      se *mod(gen,50)==0* **então**
        atualizar a utilidade π^i para cada subproblema;
      fim
    fim
  fim
fim

A aplicação de hiper-heurísticas na área de SBSE é recente, mas tem sido adotada com entusiasmo pela comunidade SBSE [31]. Contudo ainda são poucos os trabalhos que apliquem hiper-heurísticas a problemas da área de SBSE [30].

Jia et al. [32] introduziram uma hiper-heurística para aprender e aplicar estratégias de teste combinatorial. O objetivo é obter uma algoritmo genérico para aplicar este tipo de teste.

Basgalupp et al. [9] propuseram uma hiper-heurística para a geração de algoritmos que criam árvores de decisão. Estas árvores de decisão foram utilizadas na predição de esforço de software, um problema da engenharia de software que estuda como estimar o esforço necessário (expressos em horas de trabalho ou investimento financeiro) para o desenvolvimento e manutenção de software.

Os algoritmos de criação de árvores de decisão normalmente são algoritmos gulosos que recursivamente analisam um conjunto de dados que devem ou não ser particionados em um subconjunto de acordo com uma regra preestabelecida. Estes algoritmos possuem como componentes um critério de parada, um critério de partição, um método de poda (*pruning*) e um método para lidar com valores ausentes.

O critério de partição quebra os dados de entrada do nó atual da árvore de decisão. Uma regra de decisão baseada no atributo selecionado é então gerada, e os dados de entrada são filtrados de acordo com as saídas destas regras, e então o processo continua recursivamente [9]. O critério de parada determina quando a árvore de decisão deixa de ser percorrida. O método de poda é normalmente executado em árvores de decisão para reforçar a compreensão da árvore pela redução de seu tamanho mantendo (ou até melhorando) sua precisão [9]. Lidar com valores ausentes é uma tarefa importante pois valores ausentes podem ser um problema durante a avaliação realizada pelo critério de partição e durante a classificação.

Em [9], algoritmos sao tratados como LLHs. Os algoritmos são usados na criação de árvores de decisão, e podem ser formados por um dos 15 critérios de partição, 5 critérios de parada, 105 métodos para trabalhar com valores vazios (5 para *split*, 7 para distribuição e 3 para classificação) e 5 diferentes métodos de poda.

Foi realizado um experimento utilizando 5 anos de dados de um projeto de manutenção de uma companhia de TI e os resultados mostraram que a hiper-heurística gerou algoritmos que obtiveram melhores resultados do que os algoritmos tradicionais e algoritmos evolutivos.

Kumari et al. [37] propuseram uma hiper-heurística para trabalhar com o problema de agrupamento de módulos, utilizando doze heurísticas de baixo nível em um algoritmo multiobjetivo. Este problema tem como objetivo principal obter uma boa estrutura modular de um software, já que boas estruturas modulares melhoram a compreensão do software e assim facilitam o desenvolvimento e a manutenção do mesmo.

O problema de agrupamento de módulos pode ser visualizado como um grafo, onde os módulos são representados como nós e os relacionamentos como arestas. Estas arestas podem ser valoradas representando a força do relacionamento entre os módulos. Os autores utilizaram o problema representado como um vetor, onde o índice do vetor representava o módulo, o conteúdo, e o agrupamento a qual este módulo pertencia.

Em [37] a abordagem multiobjetivo foi empregada com o objetivo de obter bons agrupamentos com boa coesão e baixo acoplamento. Isto foi realizado utilizando a abordagem proposta em [45], onde os objetivos otimizados foram: maximização do somatório de intra-arcos de todos os agrupamentos, minimização do somatório de inter-arcos de todos os agrupamentos, maximização do número de agrupamentos, maximização da qualidade de modularização, minimização do número de agrupamentos isolados.

A hiper-heurística proposta trabalha com o sistema *reinforcement learning* [33], onde inicialmente as heurísticas de baixo nível são iniciadas com pesos iguais. A cada vez que uma heurística de baixo nível é selecionada seu peso é atualizado de forma que melhoras na população aumentem este peso e pioras diminuam. Esta hiper-heurística trabalha em duas fases. Na primeira fase é selecionada aleatoriamente, com igual chance, uma das mutações (troca ou cópia). Na segunda fase, utilizando informações do *reinforcement learning* e qual mutação foi escolhida, uma roleta é executada para selecionar qual das heurísticas de baixo nível será aplicada.

A seleção pode ser *rand*, que seleciona aleatoriamente os pais ou *rand-to-best*, que seleciona o melhor indivíduo da população como um dos pais, e seleciona aleatoriamente o outro pai.

A recombinação pode ser do tipo *uniform crossover*, onde os filhos são gerados pela escolha aleatória dos genes dos pais, *hybrid crossover 1* que combina *uniform crossover* com o *single-point crossover* e o *hybrid crossover 2* que combina *uniform crossover* com o *two-point crossover*.

A mutação pode ser do tipo "copia", onde dois genes são selecionados aleatoriamente e o segundo gene é copiado para o primeiro, ou tipo "troca", onde dois genes selecionados aleatoriamente trocam suas posições.

O algoritmo proposto foi aplicado em seis problemas reais e obteve melhores resultados em relação à abordagens que não usam o conceito de hiper-heurística em sua implementação .

No trabalho de Carvalho et al. [15] foi proposta a hiper-heurística MOCAITO-HH (*MOCAITO using Hyper Heuristics*), criada para trabalhar com o problema de estabelecer uma sequência de módulos para o teste e integração (*Integration and Test Order Problem* (ITO)). Nesta hiper-heurística as meta-heurísticas NSGA-II, SPEA2 e IBEA (*Indicator Based Evolutionary Algorithm*) [56] foram tratadas como operadores a serem escolhidos, e o algoritmo de seleção CF foi empregado na escolha de qual operador deveria ser aplicado a cada n gerações durante o processo evolutivo. Na MOCAITO-HH o operador selecionado recebia a população de soluções vigente, era executado, e ao final de sua execução retornava para a hiper-heurística a população de soluções trabalhada. Os resultados mostraram que a MOCAITO-HH é capaz de obter resultados equivalentes estatisticamente ao melhor algoritmo, sem que seja necessário realizar experimentos para determinar qual o mais adequado em cada sistema.

Em [28, 29] foi proposta a hiper-heurística HITO (*Hyper-heuristic for the Integration and Test Order Problem*), também criada para trabalhar com o problema ITO. Esta hiper-heurística é o principal trabalho relacionado e será melhor descrita no Capítulo 3.

2.5 Considerações Finais

Neste capítulo foram apresentados os conceitos de meta-heurísticas e hiper-heurísticas. Com relação a meta-heurísticas, foram apresentadas meta-heurísticas que empregam o conceito de dominância Pareto, como NSGA-II e SPEA2, e meta-heurísticas baseadas em decomposição (MOEA/D e MOEA/D-DRA). Com relação a hiper-heurísticas, foram apresentados os algoritmos de seleção CF (*Choice Function*), UCB (*Upper Confidence Bound*) e FRRMAB (*Fitness Rate Rank Multi Armed Bandit*) juntamente com suas metodologias de cálculo de recompensa e função de escolha de heurísticas de baixo nível.

Nesta dissertação o método de atribuição de crédito existente no FRRMAB é utilizado em conjunto com o método de seleção empregado pela *Choice Function*, ao invés de usar o método de seleção existente no FRRMAB. Assim é proposto um novo algoritmo de seleção chamado FRRCF, no Capítulo 4. Este algoritmo foi motivado pelos bons resultados da hiper-heurística HITO quando aplicada em conjunto com o algoritmo de seleção *Choice Function*.

A meta-heurística MOEA/D-DRA e os algoritmos de seleção FRRMAB e FRRCF são usados na instancia da hiper-heurística HITO-DA, proposta neste trabalho. É valido ressaltar que os algoritmos de seleção FRRMAB e FRRCF ainda não foram aplicados em problemas da área de Engenharia de Software, e desta forma ainda não foram aplicados ao problema ITO. O problema ITO e a hiper-heurística HITO são apresentados no próximo capítulo.

CAPÍTULO 3

ESTABELECENDO SEQUÊNCIAS DE MÓDULOS PARA O TESTE DE INTEGRAÇÃO

Este capítulo tem como objetivo apresentar o problema de estabelecer uma sequência de módulos para o teste de integração (*Integration Test Order Problem* (ITO)) e abordagens para resolvê-lo. Dentre elas destaca-se a abordagem MOCAITO (*Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem*), que é uma abordagem genérica para trabalhar com o problema ITO, e a hiper-heurística HITO (*Hyper-heuristic for the Integration and Test Order Problem*), que estende a abordagem MOCAITO para trabalhar com algoritmos de seleção de LLHs.

A Seção 3.1 apresenta o problema ITO. A Seção 3.2 apresenta abordagens que trabalham com este problema usando algoritmos mono-objetivo e multiobjetivos. A Seção 3.3 apresenta a abordagem MOCAITO. A Seção 3.4 apresenta a hiper-heurística HITO, o principal trabalho relacionado a esta dissertação. Por fim a Seção 3.5 apresenta as considerações finais do capítulo.

3.1 O problema de estabelecer uma sequência de módulos para o teste de integração

A atividade de teste é geralmente realizada em fases. O teste de unidade, por exemplo, visa a testar cada módulo de software separadamente. Depois disto, estes módulos são integrados e testados. Na maioria das vezes os módulos não são integrados de uma vez. Essa integração acontece em partes e, por isso a ordem de integração de módulos pode depender da disponibilidade prévia de um outro módulo, do qual eles dependem. Desta forma, faz-se necessária a determinação da ordem com que os elementos devem ser construídos e testados, e caso necessário, a construção de *stubs*.

Um *stub* é um pseudo-recurso que emula o comportamento de um recurso inexistente, sendo este criado quando um recurso é requerido por outro, mas este ainda não foi implementado. Assim um recurso r_1 pode ser desenvolvido sem o recurso r_2. Criar um *stub* tem um custo, que deve ser minimizado através de alguma estratégia que decida qual *stub* deve ser implementado primeiro. Esta tarefa não é trivial, isto porque o custo para criar um *stub* é impactado por muitos fatores e possíveis restrições contratuais.

Na Engenharia de Software este problema acontece em diferentes contextos de desenvolvimento de software, e é conhecido como problema de estabelecer uma sequência para o teste e integração de classes (*Class Integration and Test Order Problem* (CITO)) no contexto de desenvolvimento de software Orientado a Objeto (OO) e, problema de estabelecer uma sequência para o teste e integração de classes e aspectos (*Class and Aspect Integration and Test Order Problem* (CAITO)) no contexto de desenvolvimento de software Orientado a Aspecto (OA). No contexto OO um módulo é uma classe, já no contexto OA um módulo pode ser uma classe e/ou um aspecto. Este capítulo descreve abordagens baseadas em busca para resolver o problema ITO, sendo dada maior ênfase à abordagem MOCAITO, e também descreve a hiper-heurística HITO, usada como base neste trabalho.

3.1.1 Representação do Problema

O problema ITO normalmente é representado como um grafo direcionado, onde os vértices representam os módulos, e suas arestas representam os relacionamentos entre eles. Um exemplo destes grafos é o ORD (*Object Relation Diagram*) proposto por Kung et al. [38] para o problema no contexto OO, e o ORD estendido, proposto em [48], criado para o contexto OA.

O ORD é apresentado na Figura 3.1. Nesta figura, I representa o relacionamento de herança, Ag representa agregação e As representa associação entre classes. As arestas que representam relacionamentos de associação entre as classes podem ser removidas, e as outras não, já que herança e agregação apresentam além do acoplamento de controle, acoplamento de dados e dependência de código [47].

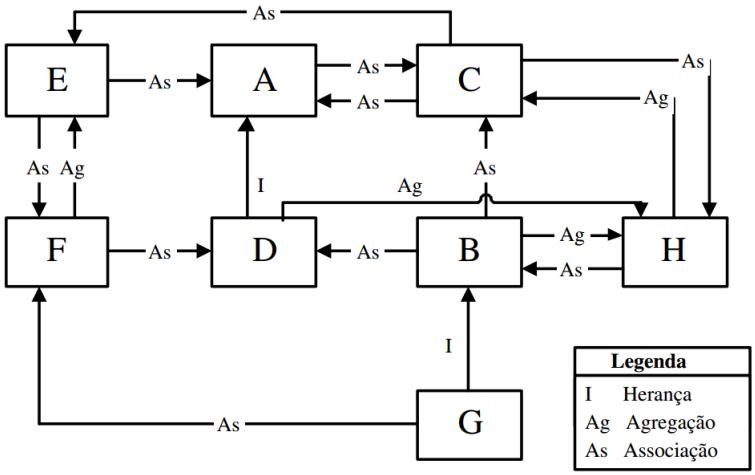


Figura 3.1: ORD (*Object Relation Diagram*), extraída de [47]

Re et al. em [48, 49] propuseram o ORD estendido para o contexto OA. O ORD estendido é mostrado na Figura 3.2. Neste grafo os vértices podem também representar aspectos, e outros tipos de relacionamentos específicos da programação orientada a aspectos, tais como uma associação de entrecorte (C), ou relacionada a um ponto de corte (As), declarações intertipo (It), etc.

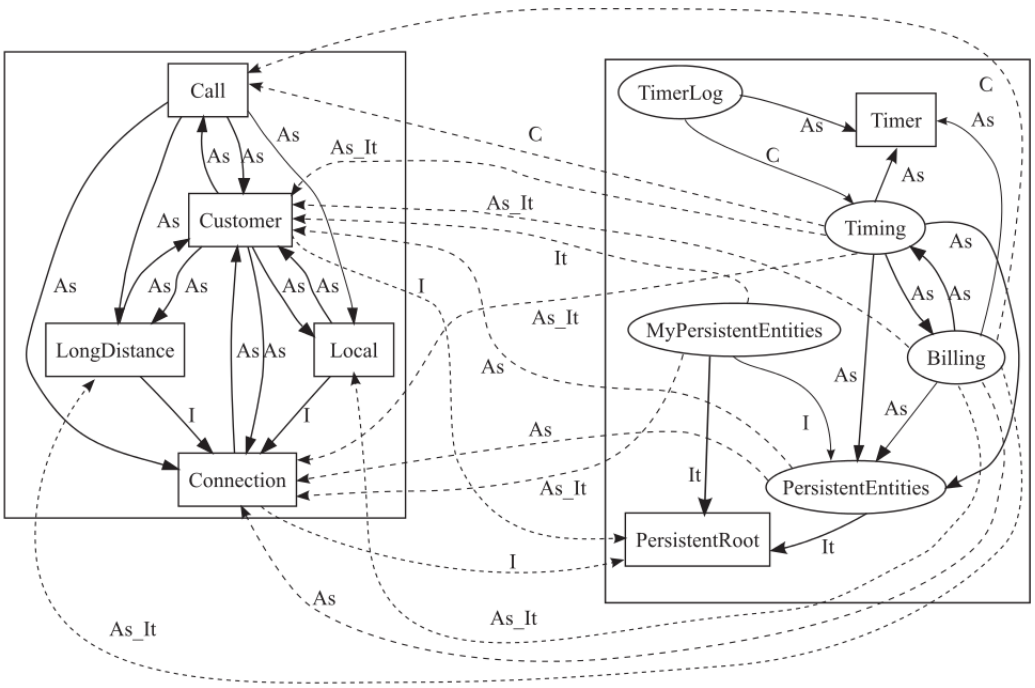


Figura 3.2: ORD Estendido, extraída de [48]

Estas representações foram usadas em [7, 28, 29] e serão usadas neste presente trabalho.

3.2 Abordagens para resolução do problema ITO

No problema ITO quando não há ciclos de dependência no grafo, a solução pode ser encontrada pela simples inversão topológica, ordenando as classes considerando suas dependências. Na maioria dos casos, esta ordenação não pode ser aplicada, devido ao fato de que a maioria dos programas contém ciclos de dependência [42], no entanto, a quebra de um ciclo de dependência no grafo implica que um *stub* deve ser construído.

Muitos trabalhos para solucionar o problema ITO usam algoritmos baseados em técnicas de grafos para determinar a melhor sequência de módulos que esteja associada ao menor custo. Estes algoritmos resolvem o problema tentando encontrar e quebrar ciclos de dependência entre os módulos.

Uma limitação observada nos algoritmos baseados em grafos é que a maioria deles identifica recursivamente componentes fortemente conectados no grafo, e em cada um destes remove uma dependência que maximiza o número de ciclos quebrados [11]. Estes algoritmos otimizam a decisão sem determinar as consequências no resultado final e em muitos casos, as soluções obtidas são sub-ótimas [7]. Outra desvantagem é que estes algoritmos são muito difíceis de ser adaptados considerando muitos fatores que podem estar envolvidos com a construção de um *stub*, tais como o número de chamadas ou métodos distintos, restrições relacionadas a razões organizacionais ou contratuais, etc [49].

Para resolver estes problemas algoritmos de busca são mais adequados. Dentre estes destacam-se algoritmos monoobjetivos, algoritmos multiobjetivos e hiper-heurísticas. A Figura 3.3 apresenta alguns estudos realizados utilizando abordagens baseadas em busca.

3.2.1 Abordagens Monoobjetivos

Briand et al. [11, 12] propuseram o uso de Algoritmos Genéticos para o contexto OO, estes algoritmos utilizavam uma função de agregação entre os objetivos número de atributos e o número de métodos, atribuindo assim um peso de 0.5 para cada objetivo.

No contexto de OA o trabalho de Galvan et al. [26] também utilizou Algoritmos Genéticos utilizando os mesmos objetivos, função de agregação e pesos usados em [11, 12].

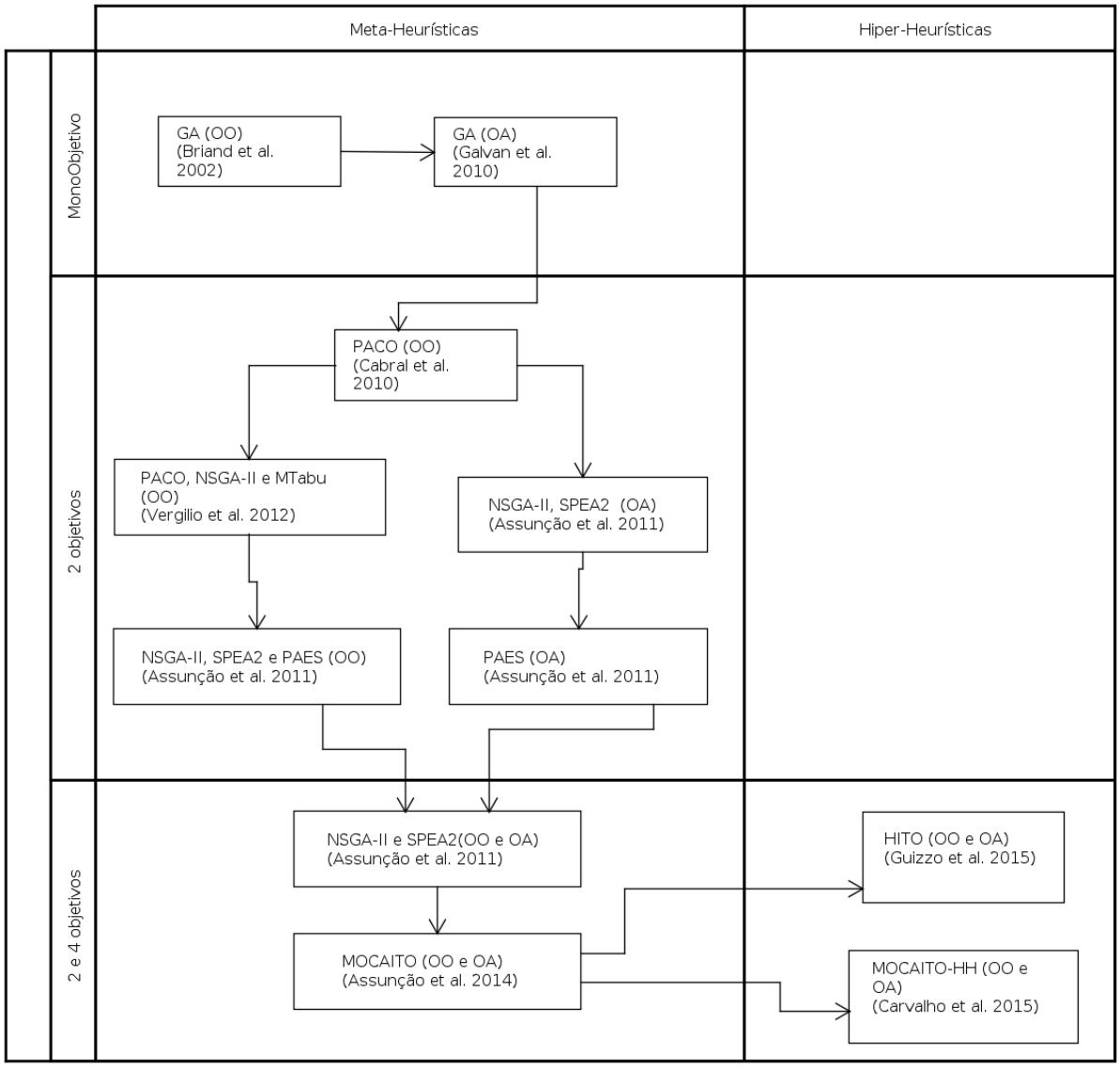


Figura 3.3: Linha de tempo dos trabalhos baseados em busca

Os resultados obtidos foram superiores aos resultados dos algoritmos baseados em grafos, entretanto o uso de uma média ponderada impõe a especificação de cada peso usado para cada objetivo. Assim o sucesso da abordagem de média ponderada depende da escolha adequada dos pesos para os objetivos, que geralmente não é um processo simples do ponto de vista do tomador de decisão [1]. Desta forma a abordagem multiobjetivo torna-se mais interessante.

3.2.2 Abordagens Multiobjetivo

Uma abordagem multiobjetivo foi introduzida no contexto OO por Cabral et al. [14] usando o conceito de fronteira de Pareto [44] para lidar com o problema CITO, neste trabalho foi utilizada a Otimização de Colônia de Formigas para problemas multiobjetivo (PACO) utilizando em sua função objetivo os dois objetivos usados em [11, 12]. Os resultados mostraram que a abordagem multiobjetivo foi superior às abordagens que utilizavam a função de agregação. Além disso, este tipo de abordagem não necessita do ajuste de pesos e ainda retorna um conjunto de boas soluções, dentre os quais o testador pode selecionar aquela que melhor lhe convier.

No trabalho de Vergilio et al. [51], os algoritmos PACO, NSGA-II e MTabu (*Multi-Objetive Tabu Search*, algoritmo multiobjetivo baseado em Busca Tabu) foram utilizados em experimentos no contexto OO. Neste trabalho a função de *fitness* foi composta pelos dois objetivos usados em [11, 12]. O algoritmo NSGA-II obteve os melhores resultados.

Ainda no contexto OO o trabalho de Assunção et al. [3] abordou este problema usando os algoritmos evolutivos NSGA-II, SPEA2 e PAES utilizando os objetivos empregados em [11, 12] e realizando o experimento com sistemas maiores e mais complexos, caracterizando assim uma aplicação mais prática. Neste experimento o PAES obteve os melhores resultados.

Com relação ao contexto de OA, Colanzi et al. [4] aplicaram os algoritmos NSGA-II e SPEA2 considerando os objetivos empregados em [11, 12], o resultado obtido apontou o NSGA-II como o melhor algoritmo. Posteriormente Assunção et al. [6] estenderam este trabalho e aplicaram o algoritmo PAES, obtendo assim melhores resultados que o trabalho anterior.

Assunção et al. [5] aplicaram os algoritmos SPEA2 e NSGA-II, usaram os objetivos propostos em [11, 12], e adicionaram dois novos objetivos: número de tipos de retornos e número de tipos distintos de parâmetros. Os resultados apontaram que ambos algoritmos obtiveram boas soluções, contudo o algoritmo SPEA2 convergiu levemente mais rápido do que o algoritmo NSGA-II.

Posteriormente a abordagem MOCAITO foi definida [7], esta abordagem permite o uso de diferentes métricas e meta-heurísticas, fazendo desta uma abordagem genérica e abrangente, que pode ser utilizada em diferentes contextos. A abordagem MOCAITO é utilizada é apresentada a seguir.

3.3 Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem (MOCAITO)

A abordagem MOCAITO (*Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem*) foi introduzida em Assunção et al. [7] com o intuito de trabalhar com o problema ITO, considerando diferentes tipos de módulos a serem integrados em diferentes contextos de software.

A abordagem MOCAITO inclui uma série de etapas distintas, onde cada etapa produz um artefato, que é usado como entrada para outras etapas (Figura 3.4).

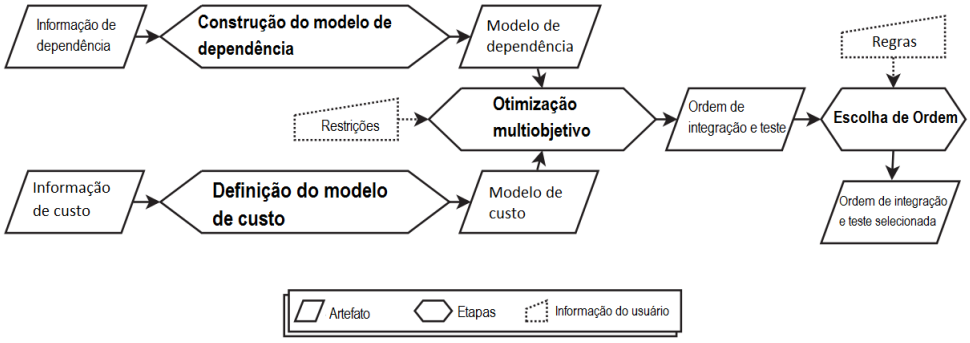


Figura 3.4: Fluxo da MOCAITO, adaptada de [7]

Esta abordagem usa duas entradas. A primeira é a informação de dependência entre os módulos que é uma representação para os relacionamentos de dependência (entre classes, entre aspectos, entre classes e aspectos, e entre componentes) [7]. A segunda entrada é a informação de custo, que é usada para calcular o custo envolvido na construção de um *stub*, seus valores são utilizados pela função de aptidão do algoritmo [7].

Outras entradas podem ser especificadas pelo testador, estas entradas são restrições que são relacionadas a imposições do ambiente e do contexto de desenvolvimento de software, ou outras restrições com o intuito de facilitar a atividade de teste [7].

Desta maneira, na etapa chamada "Construção do modelo de dependência", o modelo de dependência é construído levando em consideração o contexto do software (OO ou OA) usando alguma forma de representação. Já a etapa "Construção do modelo de custo", o modelo de custo é construído levando em consideração quais objetivos específicos devem ser usados nos algoritmos multiobjetivos.

Então a fase de "Otimização multiobjetivo" inicia usando as restrições especificadas, o modelo de custo e o modelo de dependência como entradas para um algoritmo multiobjetivo. A saída da otimização multiobjetivo é um conjunto de soluções representando a ordem de teste que têm o melhor *trade-off* entre os objetivos.

Então na etapa de "Escolha de uma ordem", o testador escolhe uma ordem a ser utilizada a partir de um conjunto de soluções não dominadas. Esta escolha deve ser realizada considerando restrições e prioridades relacionadas ao desenvolvimento de software, tais como metas, recursos disponíveis, e restrições contratuais [7].

Em [7] a abordagem MOCAITO foi instanciada e avaliada. Alguns aspectos de implementação desta abordagem são detalhados a seguir.

3.3.1 Modelo de Dependência

A implementação para a abordagem MOCAITO usou duas formas de representação para o modelo de dependência. Quando o software a ser trabalhado era Orientado a Objeto o modelo de dependência usado foi baseado no ORD (Figura 3.1). Quando o problema a ser trabalhado era Orientado a Aspecto então o modelo de dependência usado foi o ORD estendido (Figura 3.2).

3.3.2 Modelo de Custo

O modelo de custo foi composto pelas funções objetivo propostas em [5, 11, 12]. Estas funções são calculadas considerando que (i) m_i e m_j são módulos acoplados, (ii) um

módulo pode ser tanto uma classe quanto um aspecto, (iii) o termo "operação" pode representar um método de uma classe ou um método/adendo de um aspecto, e (iv) m_i depende de m_j. A seguir são apresentados as funções objetivo empregadas na instância da MOCAITO em [7], onde n representa a quantidade de módulos existentes no sistema trabalhado.

Número de Atributos (A): representa o número de atributos declarados localmente no módulo m_j que são referenciados pelo módulo m_i através de algum uso, como um tipo de retorno ou como parâmetros locais de alguma operação de m_i. Esta medida conta o (máximo) número de atributos que deveriam ser manipulados na construção de um *stub* caso a dependência fosse quebrada. Em caso de herança não são contabilizados os atributos herdados. A Equação 3.1 representa o somatório do número de atributos referenciados entre os módulos.

$$A(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} AM(i,j); j \neq k \tag{3.1}$$

Número de Operações (O): representa o número de operações (incluindo construtores) localmente declaradas em m_j que são invocadas por operações de m_i. Esta medida conta o número de operações que deverão ser emuladas em um *stub* caso a dependência seja quebrada. Em caso de herança são também contabilizadas as operações herdadas. A Equação 3.2 representa o somatório do número de operações referenciados entre os módulos.

$$O(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} OM(i,j); j \neq k \tag{3.2}$$

Número de tipos distintos de retorno (R): representa o número de tipos distintos de retorno declarados localmente em m_j que são chamados por operações em m_i. Retornos do tipo *void* não são contabilizados. Em caso de herança são contabilizados o número de tipos distintos de retornos das operações declaradas no módulo pai. A Equação 3.3 representa o somatório do número de tipos distintos de retorno referenciados entre os módulos.

$$R(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} RM(i,j); j \neq k \tag{3.3}$$

Número de tipos distintos de parâmetro (P): corresponde ao número de parâmetros declarados localmente em m_j e invocados por operações em m_i. Quando existe sobrecarga de operações, o número de parâmetros é igual a soma de todos os parâmetros distintos entre todas as implementações de cada método. Sempre se considera que existe a possibilidade de invocação de todas as implementações de determinada operação. Em caso de herança conta-se o número de tipos distintos de parâmetros das operações declaradas no módulo pai. A Equação 3.4 representa o somatório do número de tipos distintos de parâmetro referenciados entre os módulos.

$$P(t) = \sum_{i=1}^{n} \sum_{j=1}^{n} PM(i,j); j \neq k \tag{3.4}$$

3.3.3 Representação da população

Na implementação da abordagem MOCAITO foram utilizados os algoritmos NSGA-II, SPEA2 e PAES, pertencentes ao framework *JMetal* [23], escrito na linguagem Java. Para isto o problema foi modelado de forma que cada individuo da população seja representado como um vetor de inteiros onde cada posição representa um módulo através de um número identificador, onde o tamanho do vetor é igual a quantidade de módulos disponíveis. A ordem do vetor de inteiros representa a ordem que os módulos devem ser integrados.

Quando novos indivíduos são gerados, estes indivíduos não podem quebrar dependências de herança e declarações intertipos. Assim uma verificação é executada a fim de garantir a integridade dos indivíduos. A estratégia de tratamento das restrições envolve uma varredura do inicio ao fim do cromossomo, verificando se já apareceram as dependências para os módulos que são dependentes de outros módulos [7]. Caso uma restrição de precedência de módulos seja quebrada, este módulo é colocado no fim do cromossomo e todos módulos posteriores têm sua posição decrementada um [7].

Os novos indivíduos são gerados pelos operadores *Swap mutation* e *Two Point Crossover* descritos a seguir.

3.3.4 Operador de Mutação

O operador de mutação utilizado é o *Swap mutation* (Figura 3.5), que seleciona dois genes para serem trocados de posição.

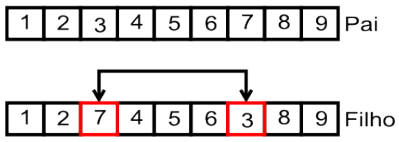


Figura 3.5: Operador de mutação *Swap mutation*, extraída de [2]

Esta Figura mostra o operador de mutação executado através da troca dos elementos 7 e 3.

3.3.5 Operador de Cruzamento

O operador de cruzamento utilizado é o *Two Point Crossover* (Figura. 3.6), onde dois pontos de corte são selecionados e através destes uma combinação é realizada para gerar dois filhos.

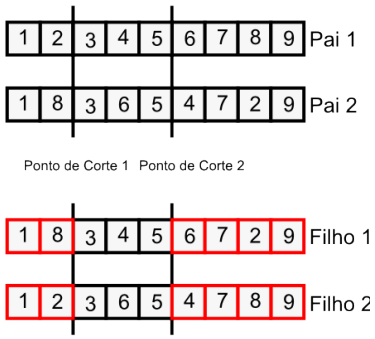


Figura 3.6: Operador de cruzamento *Two Point Crossover*, extraída de [2]

Esta Figura mostra o operador de cruzamento separando os pais em 3 partes. A primeira ($\{1, 2\}$) e terceira ($\{6, 7, 8, 9\}$) parte do pai_1 são combinadas com a segunda parte do pai_2 ($\{3, 6, 5\}$) para gerar o $filho_2$. O $filho_1$ é gerado através da combinação da primeira ($\{1, 8\}$) e terceira parte ($\{4, 7, 2, 9\}$) do pai_2 com a segunda parte do pai_1 ($\{3, 4, 5\}$).

3.4 Hyper-heuristic for the Integration and Test Order Problem (HITO)

A HITO (*Hyper-heuristic for the Integration and Test Order Problem*) [28] é uma hiper-heurística de aprendizado online para a seleção de LLHs perturbativas (operadores de mutação e cruzamento). O objetivo principal desta hiper-heurística é buscar selecionar a melhor LLH durante o processo de busca para o problema ITO.

A hiper-heurística HITO foi projetada para ser genérica e ser usada em conjunto com diferentes algoritmos multiobjetivos e algoritmos de seleção. A Figura 3.7 mostra o funcionamento da HITO.

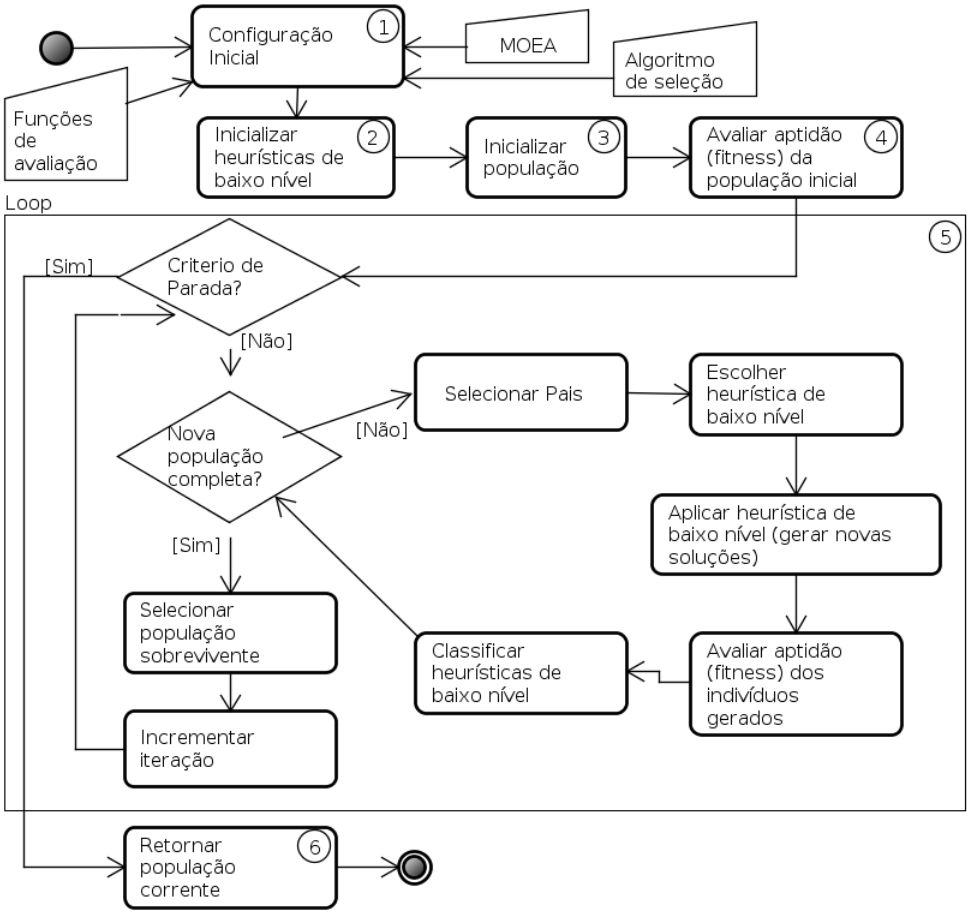


Figura 3.7: Fluxo da HITO, adaptada de [28]

A HITO inicia seu fluxo de trabalho recebendo parâmetros relacionados a utilização do algoritmo, a representação do problema empregada, a função de *fitness* utilizada, e o algoritmo de seleção de heurísticas de baixo nível a ser usado. Em seguida a HITO

inicializa as LLHs, gera uma população aleatória, e calcula o *fitness* para cada indivíduo desta população.

Após o quarto passo, é iniciado um laço que é repetido até que o critério de parada seja atingido, que neste caso é quantidade de gerações. Posteriormente um novo laço é iniciado com a função de compor uma nova população, enquanto esta população não é completada, a cada iteração, pais são selecionados através de torneio binário e usados por uma LLH, selecionada através do algoritmo de seleção, para gerar novos indivíduos. Estes novos indivíduos são então avaliados segundo a função de avaliação para assim verificar a qualidade resultante da aplicação da LLH. A atualização dos valores de qualidade para a LLH recém aplicada implica em uma nova classificação das LLHs segundo o critério empregado pela algoritmo de seleção instanciado.

O processo segue até completar o tamanho da nova geração para que seja selecionada a população sobrevivente entre a população geradora e a população gerada. A HITO finaliza seu fluxo ao atingir a quantidade máxima de iterações.

Em [28] a HITO foi estudada em conjunto com a meta-heurística NSGA-II e com as funções de seleção CF e UCB (apresentadas no Capítulo 2) e a métrica r, proposta em [28], como métrica de avaliação. Esta métrica tem seu valor obtido pela Equação 3.5.

$$r_{llh} = \frac{1}{|P|.|C|} \cdot \sum_{p \in P} \sum_{c \in C} \left\{ \begin{array}{l} 1 \text{ se } p \prec c \\ 0 \text{ se } c \prec p \\ 0.5 \text{ outro caso} \end{array} \right\} \tag{3.5}$$

onde c representa o filho; C é o conjunto de filhos gerados; p é o pai; e P é o conjunto de pais usados na geração de filhos.

O valor da métrica r é obtido através do somatório dos valores retornados pela comparação entre as soluções filhos e pais. Nesta comparação, quando um filho domina um pai através do critério de dominância de Pareto ($c \prec p$), o valor de dominância obtido é igual a 1, quando um pai domina um filho ($p \prec c$), o resultado obtido é 0, indicando que não houve melhora. Quando o filho não domina o pai e nem o pai domina o filho,

Tabela 3.1: LLHs utilizadas

		Mutação	
Cruzamento	**-**	**Swap**	**Insertion**
Two Points (2P)	h1	h2	h3
Uniform	h4	h5	h6
PMX	h7	h8	h9

o valor de dominância é igual a 0.5. Assim, se os filhos gerados por uma LLH dominam todos os pais, então $r_{llh} = 1$. Se os pais dominam todos seus filhos, então $r_{llh} = 0$. Se os filhos gerados são não dominados em relação a seus pais e vice-versa, então $r_{llh} = 0.5$. Se metade dos filhos dominam os pais e a outra metade é dominada, então $r_{llh} = 0.75$.

As LLHs usadas por Guizo et al são apresentadas na Tabela 3.1, onde foram usadas combinações dos operadores de cruzamento *Two Points Crossover*, *Uniform Crossover* e *PMXCrossover*, e os operadores de mutação *SwapMutation* e *SimpleInsertionMutation*.

Os resultados mostraram que a HITO obteve resultados superiores à instância da abordagem MOCAITO usando o NSGA-II [7] e que os uso do algoritmo de seleção *Choice Function* obteve melhores resultados do que o MAB.

Posteriormente, em [29] a HITO foi aplicada em conjunto com a meta-heurística SPEA2 com as mesmas funções de seleção, métrica de avaliação, sistemas e conjunto de LLHs usadas em [28]. Os resultados deste trabalho apontaram que a HITO em conjunto com o SPEA2 não obteve melhores resultados do que a HITO em conjunto com o NSGA-II, mas obteve resultados superiores a instância da abordagem utilizando algoritmos tradicionais.

3.5 Considerações Finais

Este capítulo descreveu o problema ITO, e diversas abordagens para solução deste problema, dentre as quais a abordagem MOCAITO e a hiper-heurística HITO. A MOCAITO tem como característica trabalhar com algoritmos multiobjetivos em fases bem definidas, e a HITO possui como característica englobar todas características da MOCAITO, mas tra-

balhando com algoritmos de seleção na escolha de LLHs que mais contribuam na geração de soluções filhos melhores que soluções pais.

A abordagem de hiper-heurística HITO foi escolhida por trabalhar com o problema ITO como base para a criação de uma nova abordagem de hiper-heurística chamada HITO-DA proposta nesta dissertação. Esta abordagem adapta a HITO para trabalhar com algoritmos de decomposição, como o MOEA/D-DRA. Isto foi motivado pelo desempenho do MOEA/D-DRA no *CEC 2009 MOEA Contest*, onde obteve o primeiro lugar [54].

A HITO-DA foi modelada para trabalhar em conjunto com o algoritmo de seleção FRRMAB (apresentado no Capítulo 2). Além disso, também foi proposto o algoritmo de seleção FRRCF, que combina o método de *Credit Assignment* utilizado pelo FRRMAB com o algoritmo de seleção CF. A hiper-heurística HITO-DA e o algoritmo de seleção FRRCF, propostos neste trabalho, são apresentados no próximo capítulo.

CAPÍTULO 4

HIPER-HEURÍSTICA BASEADA EM DECOMPOSIÇÃO PARA ESTABELECER UMA SEQUÊNCIA DE MÓDULOS PARA O TESTE DE INTEGRAÇÃO

Este capítulo tem como objetivo apresentar a hiper-heurística HITO-DA (*Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach*), que é uma hiper-heurística derivada da HITO, e trabalha com algoritmos de decomposição. Desta forma, a HITO-DA pode trabalhar em conjunto com a meta-heurística MOEA/D-DRA (apresentada no Capítulo 2).

Devido ao fato do algoritmo de seleção *Choice Function* em [28] ter obtido bons resultados em conjunto com a HITO, neste capítulo é proposto o algoritmo de seleção FRRCF, uma variação da *Choice Function* que usa os valores de FRR propostos em [40] como termo de qualidade na equação de seleção.

A Seção 4.1 apresenta a hiper-heurística HITO-DA, a Seção 4.2 apresenta o algoritmo de seleção FRRCF, a Seção 4.3 apresenta a combinação do algoritmo de seleção FRRCF com o algoritmo MOEA/D, por fim a Seção 4.4 apresenta as considerações finais do capítulo.

4.1 Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach (HITO-DA)

Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach (HITO-DA) incorpora algumas características da hiper-heurística HITO [28] para trabalhar com algoritmos de decomposição. A Figura 4.1 mostra o fluxo da HITO-DA. A principal diferença em relação a HITO é não trabalhar com a substituição de populações, mas sim de indivíduos, pois algoritmos de decomposição como o MOEA/D-DRA sempre

mantêm um tamanho fixo de subproblemas, e não realizam substituição de população, mas sim de soluções associadas a subproblemas.

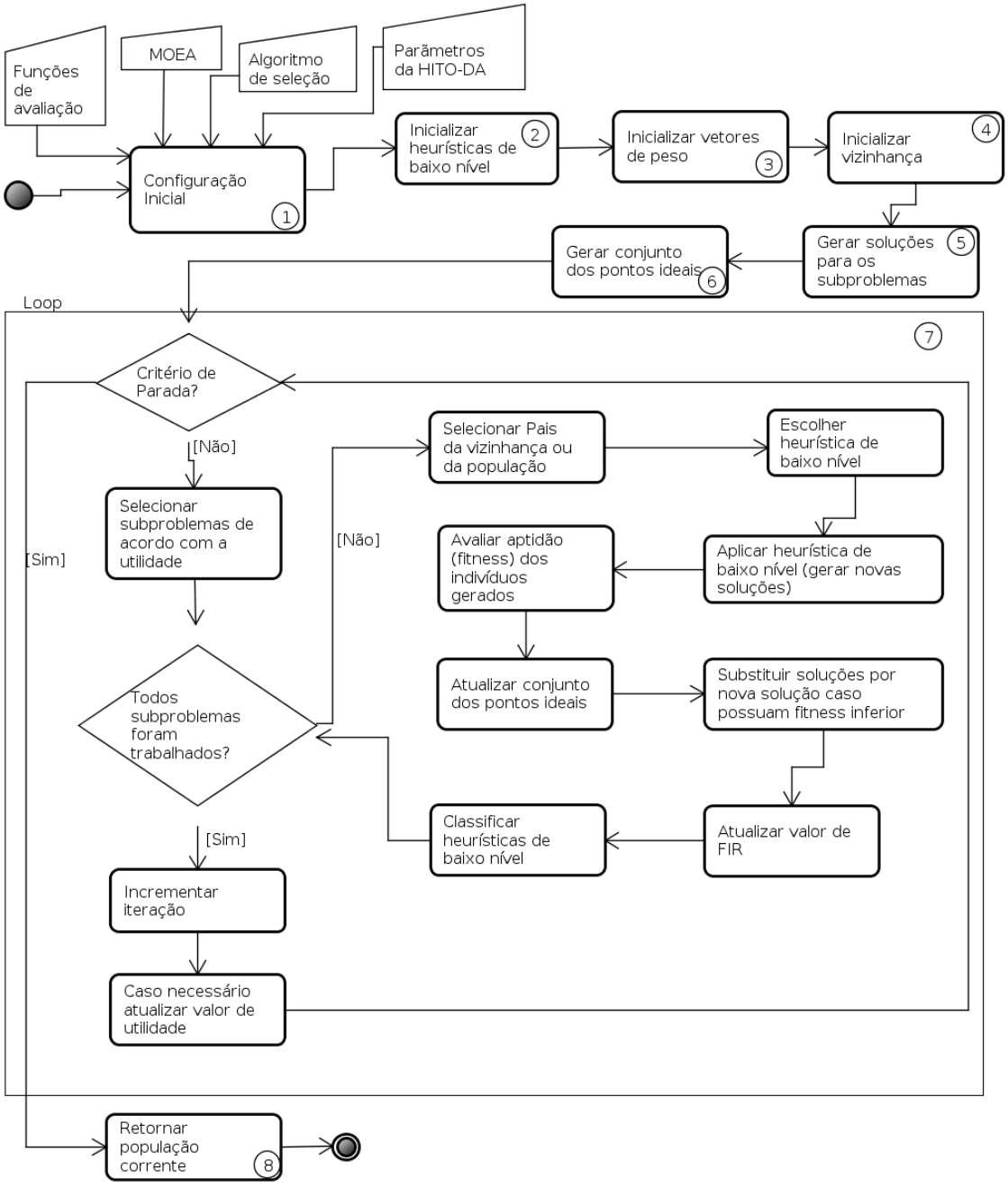


Figura 4.1: Fluxo da HITO-DA

A HITO-DA recebe como parâmetros a função de avaliação (como por exemplo a função FIR), a meta-heurística multiobjetivo (MOEA), o algoritmo de seleção (como FRRMAB e FRRCF) e parâmetros da HITO-DA (como tamanho da vizinhança e quantidade de subproblemas). Em seguida a HITO-DA inicializa as LLHs, gera os vetores de

peso, inicializa a vizinhança (baseando-se na distância Euclidiana entre os pesos), gera soluções aleatoriamente e as associam aos subproblemas, por fim, obtém o conjunto de pontos ideais.

Após o sexto passo, é iniciado um laço que é repetido até que o critério de parada seja atingido, que neste caso é quantidade de gerações. Posteriormente é obtido o conjunto de subproblemas a serem trabalhados (baseando-se em sua utilidade) e um novo laço é iniciado, e enquanto todos subproblemas não são trabalhados, para cada subproblema, pais são selecionados e usados por uma LLH, selecionada através do algoritmo de seleção, para gerar novos indivíduos. Estes novos indivíduos são então avaliados segundo a função de avaliação para assim verificar a qualidade resultante da aplicação da LLH. Em seguida o conjunto dos pontos ideais é atualizado (caso a solução gerada tenha algum valor objetivo melhor do que os existentes no conjunto dos pontos ideais), é realizada a substituição de soluções para o subproblema e atualização do valor de FIR (caso a nova solução possua melhor valor de *fitness*). A atualização dos valores de FIR para a LLH recém aplicada implica em uma nova classificação das LLHs segundo o critério empregado pela algoritmo de seleção instanciado.

O processo segue até que todos subproblemas selecionados sejam trabalhados. Após isto, a iteração é incrementada e caso necessário o valor de utilidade dos subproblemas é recalculado. A HITO-DA finaliza seu fluxo ao atingir a quantidade máxima de iterações.

Neste trabalho a HITO-DA usa como MOEA o MOEA/D-DRA (descrito na Seção 2.1), e duas diferentes funções de seleção de LLHs. O primeiro algoritmo de seleção de LLHs é o FRRMAB (descrito na Seção 2.3.3), que já foi aplicado em conjunto com a meta-heurística MOEA/D-DRA para outros problemas em [40]. O segundo algoritmo de seleção de LLHs é o FRRCF, proposto neste trabalho, que é descrito na Seção 4.2. Assim como na HITO, na HITO-DA uma LLH pode ser a combinação de um operador de mutação e outro de cruzamento, ou apenas um operador de cruzamento.

4.2 Fitness Rate Rank with Choice Function (FRRCF)

O FRRCF (*Fitness Rate Rank with Choice Function*) foi proposto devido ao fato de que a *Choice Function* obteve melhores resultados para o problema ITO ao ser aplicada em conjunto com a hiper-heurística HITO [28]. O FRRCF é um algoritmo de seleção que assim como o FRRMAB possui as etapas de escolha de LLHs e atribuição de crédito (*Credit Assignment*). No entanto, ambas as etapas foram adaptadas para trabalhar com a equação de seleção empregada na *Choice Function*. Contudo, o uso de valores FRR foi mantido na etapa de atribuição de crédito. Ambas as etapas de seleção de LLH e atribuição de crédito são descritas a seguir.

4.2.1 Atribuição de Crédito

A etapa de atribuição de crédito do FRRCF é diferente da empregada no FRRMAB, pois na *Choice Function* não é empregado o conceito de janela de tempo. Assim, ao invés de usar o vetor de *Reward* (obtido através do somatório dos valores existentes na janela de tempo), os valores de FIR são usados diretamente. Espera-se com isto que o FRRCF detecte de forma mais rápida mudanças no desempenho das LLHs.

Posteriormente, com os valores do vetor de FIR são obtidos os valores de *Rank* para cada LLH. Este valor é dado pelo ranqueamento decrescente do vetor FIR, onde a LLH que possui o maior valor no vetor FIR recebe $Rank = 1$ e a LLH com o menor valor recebe $Rank = K$.

Com os valores de FIR e *Rank* é possível obter o valor de *Decay* para cada LLH através da Equação 4.1. O valor de *Decay* é necessário para o cálculo do FRR, usado pelo método de seleção. O valor de $D \in [0, 1]$ é um parâmetro que regula a Equação 4.1, onde valores menores de D aumentam a influência da melhor LLH, resultando em escolhas mais elitistas.

$$Decay_{op} = D^{Rank_{op}} * FIR_{op} \tag{4.1}$$

Em seguida o valor de *DecaySum* é calculado através da Equação 4.2, este valor é o somatório dos valores de *Decay* de todas LLHs.

$$DecaySum = \sum_{op=1}^{K} Decay_{op} \tag{4.2}$$

A partir disto, o cálculo do valor de FRR é realizado de acordo com a Equação 4.3.

$$FRR_{op} = \frac{Decay_{op}}{DecaySum} \tag{4.3}$$

O Algoritmo 7 mostra o funcionamento da etapa de atribuição de crédito para o FRRCF. Este algoritmo recebe como parâmetros o vetor de FIR, a quantidade de LLHs (K), e o parâmetro D que regula a Equação 4.1, usada no algoritmo. O algoritmo inicia classificando em ordem decrescente o vetor de FIR (Linha 4) para que cada valor do vetor tenha uma posição (1 a K) na classificação *Rank*. Com os vetores FIR e *Rank* a Equação 4.1 é executada para cada LLH (Linha 9), gerando assim o vetor *Decay*. Posteriormente, o algoritmo executa o somatório dos valores contidos no vetor *Decay* pela Equação 4.2 para que este somatório seja usado em conjunto com os valores contidos no vetor *Decay* no cálculo do FRR, este cálculo é executado para cada LLH segundo a Equação 4.3 (Linha 13). Ao final do algoritmo o vetor FRR é retornado.

Algoritmo 7: pseudocódigo da atribuição de crédito do FRRCF

```
1  Entrada: FIR, D, K
2  Saída: FRR
3  início
4  |  Classificar vetor de FIR em ordem decrescente;
5  |  para op ← 1 até K faça
6  |  |  Rank_op = FIR.getOpPosition(op);
7  |  fim
8  |  para op ← 1 até K faça
9  |  |  Decay_op = D^Rank_op * FIR_op;
10 |  fim
11 |  DecaySum = Σ_{op=1}^{K} Decay_op;
12 |  para op ← 1 até K faça
13 |  |  FRR_op = Decay_op/DecaySum;
14 |  fim
15 |  retorna FRR;
16 fim
```

Uma modificação foi necessária no cálculo do FIR, devido ao fato de não ser empregado o conceito de janela de tempo na *Choice Function*. Pois, caso o cálculo do FIR anterior fosse mantido, as LLHs que fossem escolhidas primeiro acumulariam um valor de FIR

muito superior ao das outras, já que no FRRMAB o valor de FIR é um somatório de n (melhora de filhos em relação aos pais). Desta forma existem algumas possíveis soluções:

- Não usar o valor de FIR como somatório, mas usar o último valor de n disponível para a LLH.
- Não usar o valor de FIR como somatório, mas usar a média de n para a LLH.
- Usar um valor representativo, assim como a abordagem HITO [28]. A Equação 4.4 mostra este método.

$$FIR_{op} = FIR_{op} + \left\{ \begin{array}{l} 1 \text{ se } \frac{(F(p)-F(c))}{F(p)} > 0 \\ \text{-1 se } \frac{(F(p)-F(c))}{F(p)} < 0 \\ 0 \text{ outro caso} \end{array} \right\} \tag{4.4}$$

Experimentos foram realizados para verificar qual a melhor alternativa para o cálculo do FIR. Dentre estas soluções a que obteve os melhores resultados nos experimentos foi a alternativa de usar um valor representativo e por isto, foi adotada neste trabalho.

4.2.2 Escolha de LLHs

A escolha de LLHs é realizada pela equação de seleção da *Choice Function* (Equação 4.5). Nesta equação a intensificação é realizada pelo uso de um FRR para cada LLH, e a exploração é realizada pelo vetor $WaitingTime$ que representa a quantidade de rodadas em que uma dada LLH não é executada. Os valores α e β são parâmetros que regulam a equação.

$$op = argmax_{i=1...K}(FRR_i * \alpha + WaitingTime_i * \beta) \tag{4.5}$$

O Algoritmo 8 mostra o funcionamento do algoritmo de seleção empregado. Este algoritmo recebe como parâmetros o vetor FRR (gerado pela etapa de atribuição de crédito), K que representa a quantidade de LLHs, o vetor $WaitingTime$ que contém a quantidade de rodadas que cada LLH está inativa, e os parâmetros α e β usados na

Equação 4.5. O algoritmo inicia com a verificação se existem LLHs ainda não selecionadas, e caso haja, uma LLH é selecionada aleatoriamente. Caso todas LLHs ja tenham sido selecionadas, a LLH que maximiza a Equação 4.5 é selecionada (Linha 8). Posteriormente, na Linha 9 uma repetição é iniciada para incrementar o $WaitingTime$ para cada LLH não escolhida. Em seguida o valor de $WaitingTime$ para a LLH selecionada recebe 0. O algoritmo retorna a LLH selecionada, o parâmetro $WaitingTime$ mantém seus valores atualizados.

Algoritmo 8: Pseudocódigo do método de seleção da FRRCF

```
1  Entrada: FRR, WaitingTime_i α, β, K
2  Saída: op
3  início
4      se Existem LLHs ainda não selecionados então
5          op=Selecionar aleatoriamente algum LLH ainda não selecionado;
6      fim
7      senão
8          op = argmax_{i=1...K}(FRR_i * α + WaitingTime_i * β);
9          para i ← 1 até K faça
10             WaitingTime_i = WaitingTime_i + 1
11         fim
12         WaitingTime_op = 0
13     fim
14     retorna op;
15 fim
```

4.3 MOEA/D-FRRCF

O MOEA/D-FRRCF aplica o algoritmo de seleção FRRCF na escolha de operadores de mutação e cruzamento para a meta-heurística MOEA/D-DRA. Assim todas as tarefas do FRRCF são executadas conforme mostrado na Seção 4.2. Para isto, o valor de *Fitness* para cada LLH é calculado segundo a Equação 4.6.

$$F = g(x^j|\lambda^j, z^*) \tag{4.6}$$

onde g é a função *Tchebycheff* $g(subproblema|\text{pesos associados}, \text{pontos ideais})$

O Algoritmo 9 mostra o funcionamento do MOEA/D-FRRCF que possui como parâmetros as variáveis α e β, usados pela CF, a variável D usada pela etapa de atribuição de crédito, e a variável K que representa a quantidade de LLHs disponíveis. Este algoritmo é uma versão do MOEA/D-DRA original [53] e desta forma incorpora todas

as tarefas executadas pelo mesmo. As diferenças entre o MOEA/D-DRA e o MOEA/D-FRRCF são o uso dos parâmetros α e β, usados pela CF e a variável K que representa a quantidade de LLHs disponíveis. Outra diferença é a escolha da LLH a ser aplicada (Linha 23), que é realizada pelo método de seleção apresentado no Algoritmo 8. Nas Linhas 29, 33 e 36 é realizado o cálculo do FIR segundo a Equação 4.4. Posteriormente é executado o cálculo de FRR usando os valores de FIR existentes na janela de tempo. Isto é realizado na Linha 40, onde o cálculo é executado pelo Algoritmo 7.

Algoritmo 9: Pseudocódigo da hiper-heurística FRRCF

```
1  Entrada: T, Nr, δ, C, D, W, K, maxGen, N
2  início
3      Inicializa a população P e os pontos ideais z*;
4      FRR ← ∅;
5      FIR ← Inicializar vetor com valor 0 para cada op;
6      WaitingTime ← Inicializar vetor com valor 0 para cada op;
7      gen = 0;
8      para i ← 1 até N faça
9          B(i) = {i_1, ..., i_T} onde λ^{i_1}, ..., λ^{i_T} são T vetores mais próximos de λ^i ;
10         π^i = 1;
11     fim
12     enquanto gen < maxGen faça
13         Formar o conjunto I levando em consideração o valor de utilidade π de cada subproblema;
14         para cada i ∈ I faça
15             se Random(0,1) < δ então
16                 SP=B(i);
17             fim
18             senão
19                 SP=P;
20             fim
21             p1 ← Selecionar aleatoriamente item de SP;
22             p2 ← i;
23             op=CFSelector(FRR, WaitingTime, α, β);
24             y ← op.apply(p1, p2);
25             Atualizar ponto de referência z*;
26             c=0;
27             enquanto c < Nr||P! =∅ faça
28                 aleatoriamente selecione uma solução x^j de SP;
29                 n = (g(x^j|λ^j,z*) − g(y|λ^j,z*)) / g(x^j|λ^j,z*);
30                 se n > 0 então
31                     Substituir x^j por y;
32                     Remover x^j de P;
33                     FIR_op = FIR_op + 1;
34                 fim
35                 senão
36                     FIR_op = FIR_op − 1;
37                 fim
38                 c++;
39             fim
40             FRR=CFCreditAssignment(FIR, D, K, WaitingTime);
41             gen++;
42             se mod(gen,50)==0 então
43                 atualizar a utilidade π^i para cada subproblema;
44             fim
45         fim
46     fim
47 fim
```

4.4 Considerações Finais

Este capítulo introduziu a hiper-heurística HITO-DA, uma hiper-heurística que estende a hiper-heurística HITO e a adapta para trabalhar com meta-heurísticas baseadas em decomposição como MOEA/D-DRA. Neste trabalho a HITO-DA é instanciada em conjunto com o algoritmo de seleção FRRMAB (apresentado no Capítulo 2) e o algoritmo FRRCF proposto nesta dissertação.

O próximo capítulo apresenta o estudo experimental realizado com a finalidade de verificar o comportamento da HITO-DA.

CAPÍTULO 5

ESTUDO EXPERIMENTAL

Este capítulo descreve os experimentos realizados com o propósito de avaliar a hiper-heurística HITO-DA. Para isto, foram elaboradas algumas questões de pesquisa, descritas na Seção 5.1. Os sistemas utilizados nos experimentos são apresentados na Seção 5.2. A Seção 5.3 mostra como os resultados foram avaliados. A Seção 5.4 descreve como os experimentos foram conduzidos. A Seção 5.5 avalia os resultados obtidos. Com base nestes resultados, as questões de pesquisa são respondidas na Seção 5.6. A Seção 5.7 discute as ameaças à validade. Por fim, na Seção 5.8 são apresentadas as considerações finais do capítulo.

5.1 Questões de pesquisa

O estudo realizado teve como objetivo avaliar o desempenho da HITO-DA e responder a duas questões de pesquisa:

RQ1: Como são os resultados da HITO-DA com relação ao algoritmo MOEA/D-DRA? Esta questão é importante pois a HITO-DA trabalha em conjunto com o MOEA/D-DRA, e desta forma é necessário verificar se a HITO-DA contribui para a melhora dos resultados, ou se pelo menos é equivalente ao MOEA/D-DRA original.

RQ2: Como são os resultados da HITO-DA com relação à hiper-heurística HITO? Esta questão tem como objetivo avaliar os resultados obtidos pela HITO-DA com os da hiper-heurística HITO. Como resultado desta comparação também foi possível avaliar a HITO-DA em comparação com as meta-heurísticas NSGA-II e SPEA2, assim como avaliar quais LLHs são mais selecionadas.

Para responder às questões de pesquisa foram executados dois experimentos para a HITO-DA nomeados MOEAD-CF e MOEAD-MAB. O experimento MOEAD-CF tem como finalidade identificar as melhores configurações obtidas para a HITO-DA, utilizando-se a meta-heurística MOEAD/D-DRA em conjunto com o algoritmo de seleção FRRCF. No experimento MOEAD-MAB, a HITO-DA utiliza a meta-heurística MOEAD/D-DRA em conjunto com o FRRMAB.

Para responder à questão **RQ1** foi também realizado o experimento MOEA/D para verificar quais são as melhores configurações para a meta-heurística MOEAD/D-DRA. Este experimento foi necessário para que os resultados das melhores configurações obtidas por instâncias da HITO-DA (MOEAD-CF e MOEAD-MAB) fossem comparados com os resultados da melhor configuração obtida para a meta-heurística MOEAD/D-DRA.

Para responder à questão de pesquisa **RQ2** foram realizados experimentos com a finalidade de encontrar as melhores configurações para a HITO. Os experimentos NSGA-II-CF e NSGA-II-MAB identificaram as melhores configurações obtidas para a HITO instanciando a meta-heurística NSGA-II, em conjunto com os algoritmos de seleção CF e MAB respectivamente. Analogamente, os experimentos SPEA2-CF e SPEA2-MAB identificaram as melhores configurações obtidas para a HITO, instanciando a meta-heurística SPEA2 em conjunto com os algoritmos de seleção CF e MAB. Estes experimentos foram necessários pois os melhores resultados encontrados foram comparados aos melhores resultados obtidos pela HITO-DA. Os experimentos NSGA-II e SPEA2 instanciaram a abordagem MOCAITO com os algoritmos homônimos.

5.2 Sistemas Utilizados

Para realização dos experimentos foram usados sete sistemas, sendo quatro sistemas OO e três sistemas OA. Estes sistemas foram os mesmos utilizados em [7, 28, 29] e estão apresentados na Tabela 5.1. Nesta tabela na segunda e terceira colunas têm-se o tipo e a versão do sistema, respectivamente. Na quarta coluna tem-se a quantidade de linhas de código, na quinta coluna a quantidade de classes e na sexta coluna a quantidade de

aspectos, caso o sistema seja do contexto OA, a sexta coluna apresenta a referência onde os sistemas foram obtidos.

Tabela 5.1: Sistemas Utilizados no Experimento

Sistema	**Tipo**	**Versão**	**LOC**	**Classes**	**Aspectos**	**Referência**
BCEL	OO	5.0	2999	45	-	http://archive.apache.org/dist/jakarta/bcel/old/v5.0/
JBoss	OO	6.0.0M5	8434	148	-	http://www.jboss.org/jbossas/downloads.html
JHotDraw	OO	7.5.1	20273	197	-	http://sourceforge.net/projects/jhotdraw/
MyBatis	OO	3.0.2 2	23535	331	-	http://code.google.com/p/mybatis/downloads/list
AJHotDraw	OA	0.4	18586	288	31	http://sourceforge.net/projects/ajhotdraw/
AJHsqldb	OA	18	68550	275	30	http://sourceforge.net/projects/ajhsqldb/
Health-Watcher	OA	9	5479	95	22	http://www.comp.lancs.ac.uk/?greenwop/tao/

Para o contexto OO foram usados os sistemas *BCEL*, *JBoss*, *JHotDraw* e *MyBatis*. O sistema *BCEL* (Byte Code Engineering Library) é uma biblioteca que permite analisar, criar e manipular arquivos binários desenvolvidos em Java. O sistema *JBoss* é um servidor de aplicações Java. O sistema *JHotDraw* é um framework de gráficos bidimensionais para editores de desenho estruturado. O *MyBatis* é um framework de mapeamento de classes para aplicações OO que utilizam bancos de dados relacionais.

Para o contexto OA foram utilizados os sistemas *AJHotDraw*, *AJHsqldb* e *Health-Watcher*. O *AJHsqldb* é um gerenciador de banco de dados, o *AJHotDraw* é uma versão mais recente do *JHotDraw* e o *Health-Watcher*, que é um sistema de coleta e gerenciamento de notificações e reclamações relacionadas à saúde publica.

5.3 Ferramentas e Indicadores de Qualidade

Para realizar a comparação de resultados obtidos por diferentes algoritmos é necessário que os algoritmos sejam executados em um determinado número n de vezes, e assim n populações finais sejam obtidas para cada algoritmo. Cada uma das n execuções gera uma fronteira de Pareto que contém as melhores soluções encontradas na execução. Estas fronteiras recebem o nome de PF_{approx}.

Posteriormente, é realizada a união de todas fronteiras PF_{approx} de uma dada instância de um experimento, e em seguida é realizada a remoção das soluções dominadas desta

união. Esta união contém portanto as melhores soluções encontradas para uma dada instância de um experimento e recebe o nome de PF_{known}.

Por fim a fronteira $PF_{trueknown}$ é obtida. Esta fronteira é obtida através da união de todas fronteiras PF_{known} obtidas em todos experimentos. Neste caso as soluções dominadas também são removidas. Desta forma a fronteira $PF_{trueknown}$ contém as melhores soluções encontradas.

Neste trabalho cada um dos algoritmos foi executado uma quantidade de vezes $n = 30$, para que posteriormente a métrica de qualidade *hypervolume* fosse usada para gerar 30 valores de *hypervolume* para cada algoritmo, um para cada PF_{known}. Para isto, as fronteiras de Pareto das populações finais são normalizadas entre 0 e 1 levando em consideração os valores objetivos de todas soluções encontradas (considerando soluções dominadas) por todas instâncias, para que assim os resultados possam ser comparáveis.

O indicador *hypervolume* computa a área (ou volume quando se trabalha com mais de 2 objetivos) do espaço de objetivos que uma fronteira de Pareto domina [58]. Este cálculo é realizado usando um dado ponto de referência, que normalmente é o pior ponto possível, onde caso as fronteiras de Pareto estejam normalizadas o pior ponto possível é dado por $(1.01, 1.01)$ para dois objetivos.

O *hypervolume* permite ao engenheiro de software determinar se uma frente de Pareto é melhor do que outra através da comparação de seus valores de *hypervolume*, onde um maior valor de *hypervolume* é considerado melhor.

Com os 30 valores de *hypervolume* é possível obter a média, o desvio padrão e realizar testes estatísticos com a finalidade de verificar possíveis diferenças nos resultados.

Para verificar diferenças estatísticas entre instâncias em um dado sistema, foi empregado o teste estatístico de Kruskal-Wallis [20] com significância de 95%. Para isto os valores de *hypervolume* obtidos pelas instâncias em um dado sistema foram comparados entre si. No entanto, o uso deste teste não auxilia na descoberta de qual é a melhor instância em termos gerais, ou seja, que obteve um bom desempenho em todos os sistemas. Para isto foi utilizado o sistema de classificação empregado no teste de Friedman [25], também com significância de 95%. Este teste foi realizado utilizando o pacote CONTROLTEST [21].

5.4 Configuração dos Experimentos

A Tabela. 5.2 apresenta os experimentos executados neste trabalho (descritos na Seção 5.1) e os conjuntos de valores utilizado para cada parâmetro. Desta forma, foram criadas diversas instâncias para cada experimento através da combinação dos elementos dos conjuntos de parâmetros.

Tabela 5.2: Configuração de experimentos

	MOEAD-MAB	**MOEAD-CF**	**NSGA-II-CF**	**NSGA-II-MAB**	**SPEA2-II-CF**	**SPEA2-II-MAB**	**MOEA/D**
Pop/Arq	300	300	300	300	300	300	300
T	30	30	-	-	-	-	30
Nr	3, 6, 30	3, 6, 30	-	-	-	-	3, 6, 30
δ	0.9	0.9	-	-	-	-	0.9
D	0.1, 0.3, 0.5, 0.7, 0.9, 1.0	0.1, 0.3, 0.5, 0.7, 0.9, 1.0	-	-	-	-	-
W	150, 300	-	-	150, 300	-	150, 300	-
C	0.5, 1.0, 2.0, 5.0	-	-	0.5, 1.0, 2.0, 5.0	-	0.5, 1.0, 2.0, 5.0	-
α	-	1	1	-	1	-	-
β	-	0.10 0.05 0.01 0.005 0.001 0.00005	0.10 0.05 0.01 0.005 0.001 0.00005	-	0.10 0.05 0.01 0.005 0.001 0.00005	-	-

Foram utilizados os dois objetivos (Número de Operações (O) e Número de Atributos (A)) descritos na Seção 3.3.2 e empregados nos trabalhos [5, 11, 12].

O tamanho da população foi obtido dos trabalhos relacionados que tratam do mesmo problema [7, 28, 29]. Os parâmetros W, D e C relativos ao MAB foram escolhidos conforme [28, 40]. Os parâmetros do MOEA/D-DRA, T e δ, foram selecionados segundo [53, 54], e o parâmetro Nr foi selecionado segundo [52]. Os valores de α e β foram selecionados de acordo com [28, 29]. Neste caso foram escolhidos também outros valores para o parâmetro β a fim de se ter uma escolha menos elitista para a *Choice Function*.

Os experimentos foram executados de forma independente durante 30 execuções, e assim gerando 30 fronteiras PF_{approx}. Posteriormente, os valores de *hypervolume* para cada instância foram calculados conforme descrito na Seção 5.3

O Apêndice A apresenta os resultados para o teste de parâmetros, e a partir da análise destes resultados foram selecionadas as melhores instâncias.

Tabela 5.3: Configurações selecionadas

Alg	Pop/Arq	T. Muta	T. Cruza	T	Nr	δ	D	α	β	C	W
MOEAD-MAB	300	1	1	30	6	0.9	0.1	-	-	5.0	150
MOEAD-CF	300	1	1	30	6	0.9	1.0	1	0.10	-	-
NSGA-II-CF	300	1	1	-	-	-	-	1	0.00005	-	-
NSGA-II-MAB	300	1	1	-	-	-	-	-	-	2.0	150
SPEA2-CF	300	1	1	-	-	-	-	1	0.00005	-	-
SPEA2-MAB	300	1	1	-	-	-	-	-	-	0.5	150
MOEA/D	300	1	1	30	3	0.9	-	-	-	-	-
NSGA-II	300	0.02	0.95	-	-	-	-	-	-	-	-
SPEA2	300	0.02	0.95	-	-	-	-	-	-	-	-

As melhores configurações de instâncias encontradas para cada experimento foram selecionadas conforme descrito na Seção 5.3. As melhores configurações de cada experimento são apresentadas na Tabela 5.3. Nesta tabela também são apresentadas as configurações dos experimentos NSGA-II e SPEA2, que são as mesmas configurações usadas em [7].

Com o propósito de responder à **RQ1**, foi realizada uma análise comparando os resultados obtidos pelas instâncias MOEAD-CF, MOEAD-MAB e MOEA/D.

Para responder à **RQ2**, foi realizada uma análise comparando os resultados obtidos pela HITO-DA (MOEAD-CF e MOEAD-MAB) com os resultados obtidos pela HITO (NSGA-II-CF, NSGA-II-MAB, SPEA2-CF, SPEA2-MAB) e pelas meta-heurísticas (NSGA-II e SPEA2). As escolhas de LLHs realizadas por configurações da HITO-DA foram coletadas e usadas para o cálculo da média de quantidade de escolhas.

5.5 Resultados

Esta seção apresenta os resultados de forma dividida. Na Subseção 5.5.1 são analisados os resultados com relação ao *hypervolume* e quantidade de soluções não dominadas. A Subseção 5.5.2 apresenta uma análise utilizando as fronteiras de Pareto geradas pe-

los algoritmos. A Subseção 5.5.3 apresenta uma análise da seleção de LLHs realizada pela HITO-DA. A Subseção 5.5.4 apresenta a análise geral dos resultados. Por fim, a Subseção 5.5.5 apresenta a análise das soluções geradas.

5.5.1 Resultados para *hypervolume* e quantidade de soluções não dominadas

As Tabelas 5.4, 5.5 e 5.6 apresentam as quantidades de soluções existentes em $PF_{trueknown}$. Nas colunas H_v são apresentadas as médias de *hypervolume* e desvio-padrão (entre parênteses). Nas colunas N_d são apresentadas as médias de quantidade de soluções existentes em PF_{approx} e as quantidades de soluções existentes em PF_{known} que também existem em $PF_{trueknown}$ (entre parênteses).

A Tabela 5.4 apresenta os resultados para as instâncias da HITO-DA comparados a instância MOEA/D. Os resultados mostraram que na maioria dos casos as médias de *hypervolume* obtidas pela HITO-DA foram maiores com diferença estatística quando comparadas com o MOEA/D-DRA. Apenas nos sistemas *JBoss* e *Health-Watcher* constata-se equivalência estatística nos resultados com menor média de *hypervolume* obtida pela instância MOEA/D.

Os resultados também mostraram que a instância MOEA/D não obteve soluções na fronteira $PF_{trueknown}$ nos sistemas *MyBatis*, *JHotDraw*, *AJHsqldb* e *AJHotDraw*. Para o sistema *BCEL*, a quantidade de soluções foi menor do que a quantidade das instâncias da HITO-DA.

Nos sistemas *JBoss* e *Health-Watcher* as três instâncias comparadas obtiveram a mesma fronteira $PF_{trueknown}$. Estes sistemas possuem apenas uma solução na fronteira $PF_{trueknown}$, ou seja, todas as instâncias testadas (HITO, HITO-DA e meta-heurísticas) obtiveram apenas uma solução não dominada em suas fronteiras PF_{known}. No sistema *JBoss*, a instância MOEA/D obteve uma média de *hypervolume* inferior a obtida pelas instâncias da HITO-DA mesmo obtendo a mesma fronteira PF_{known}. Isto se deve ao fato de que em uma das 30 execuções, a instância MOEA/D encontrou uma solução que é

Tabela 5.4: Resultados da comparação da HITO-DA com o MOEA/D

Sistema	$PF_{trueknown}$	**MOEA/D**		**MOEAD-CF**		**MOEAD-MAB**	
		H_v	N_d	H_v	N_d	H_v	N_d
MyBatis	50	0,51604 (0,07289)	36,27 (0)	**0,70966 (0,04470)**	38,80 (3)	**0,71292 (0,04501)**	37,80 (42)
BCEL	29	0,66306 (0,04655)	28,33 (22)	**0,72914 (0,00817)**	29,13 (29)	**0,73127 (0,00517)**	29,17 (29)
JHotDraw	2	0,42821 (0,24306)	1,53 (0)	**0,58827 (0,20472)**	1,57 (0)	**0,60494 (0,23102)**	1,67 (1)
JBoss	1	**0,70264 (0,46199)**	1,00 (1)	**1,00000 (0,00000)**	1,00 (1)	**1,00000 (0,00000)**	1,00 (1)
AJHsqldb	30	0,26847 (0,10769)	11,57 (0)	**0,74872 (0,07126)**	19,60 (22)	**0,78351 (0,06697)**	22,80 (5)
AJHotDraw	8	0,27691 (0,14578)	2,60 (0)	**0,64652 (0,10937)**	5,63 (0)	**0,65990 (0,16751)**	5,27 (7)
Health-Watcher	1	**0,80006 (0,31857)**	1,03 (1)	**1,00000 (0,00000)**	1,00 (1)	**0,99529 (0,02582)**	1,00 (1)

dominada pela solução existente em PF_{known}. Esta situação também ocorre no sistema *Health-Watcher* com relação às instâncias MOEA/D e MOEAD-MAB.

Comparando as instâncias MOEAD-CF e MOEAD-MAB constata-se que suas médias de *hypervolume* são equivalentes em todos os sete sistemas. A instância MOEAD-MAB obteve maior média de *hypervolume* nos sistemas *MyBatis*, *BCEL*, *JHotDraw*, *AJHsqldb* e *AJHotDraw*. A instância MOEAD-CF obteve maior média apenas no sistema *Health-Watcher*. Ambas instâncias obtiveram a mesma média de *hypervolume* no sistema *JBoss*.

As Tabelas 5.5 e 5.6 apresentam os resultados para sistemas do contexto OO e OA respectivamente. Nestas tabelas as instâncias da HITO-DA são comparadas as instâncias da HITO e as meta-heurísticas NSGA-II e SPEA.

No sistema *MyBatis* observaram-se equivalências estatísticas nos resultados de *hypervolume* entre o MOEAD-MAB, MOEAD-CF e NSGA-II-CF, onde a melhor média foi obtida pela instância MOEAD-MAB. As meta-heurísticas e outras instâncias da HITO obtiveram médias inferiores de *hypervolume* com diferença estatística. Com relação à quantidade de soluções existentes em PF_{known}, constata-se uma quantidade maior de soluções obtidas pela instância MOEAD-MAB em relação às outras.

No sistema *BCEL* nenhuma instância da HITO-DA obteve resultados equivalentes aos resultados obtidos pela HITO, neste sistema três das quatro instâncias da HITO foram equivalentes entre si. A instância NSGA-II-CF obteve a maior média de *hypervolume* para o sistema *BCEL*. Com relação à quantidade de soluções existentes em PF_{known}, constata-se a mesma quantidade (29 soluções) em quase todas as instâncias, onde apenas a instância SPEA2-CF obteve um número menor (28).

No sistema *JHotDraw* todas as instâncias foram equivalentes estatisticamente com relação ao *hypervolume*, onde a instância NSGA-II-CF obteve a maior média. Neste sistema, com relação à quantidade de soluções existentes em PF_{known}, constata-se que as instâncias NSGA-II-CF e SPEA2-CF encontraram as duas soluções existentes na fronteira $PF_{trueknown}$, enquanto a instância MOEAD-MAB encontrou uma solução, e as outras instâncias não encontraram soluções.

No sistema *JBoss* todas as instâncias foram equivalentes estatisticamente e todas instâncias obtiveram a mesma média de *hypervolume*. Neste sistema, todas instâncias encontraram a mesma solução em todas suas execuções. Desta forma, não houve desvio padrão no *hypervolume* em nenhuma das instâncias.

Com relação aos sistemas OA, no *AJHsqldb* observaram-se equivalências estatísticas nos resultados entre as instâncias MOEAD-MAB e MOEAD-CF, e cujos resultados são melhores do que os resultados das instâncias da HITO. Neste sistema a melhor média de *hypervolume* foi obtida pela instância MOEAD-MAB. Com relação à quantidade de soluções existentes em PF_{known}, a instância MOEAD-CF obteve uma quantidade maior de soluções.

No sistema *AJHotDraw*, as instâncias da HITO-DA e HITO obtiveram resultados equivalentes. A configuração NSGA-II-CF obteve a maior média de *hypervolume*. Com relação à quantidade de soluções existentes em PF_{known}, a instância MOEAD-MAB obteve uma quantidade maior de soluções.

No sistema *Health-Watcher* todos os algoritmos obtiveram médias de *hypervolume* com equivalência estatística. Neste sistema, todas instâncias destacadas (maior média de *hypervolume*) encontraram a mesma solução em todas suas execuções. A instância

Tabela 5.5: Resultados da comparação da HITO-DA com outros algoritmos para sistemas do contexto OO

Alg	**MyBatis**		**BCEL**		**JHotDraw**		**JBoss**	
$Pf_{trueKnown}$	50		29		2		1	
	H_v	N_d	H_v	N_d	H_v	N_d	H_v	N_d
NSGA-II	0,74316 (0,04679)	58,87 (0)	0,66824 (0,01069)	28,67 (29)	**0,67662 (0,20095)**	1,37 (0)	**0,25498 (0,00000)**	1,00 (1)
SPEA2	0,72504 (0,05090)	54,93 (0)	0,66919 (0,01121)	28,83 (29)	**0,58732 (0,23660)**	1,40 (0)	**0,25498 (0,00000)**	1,00 (1)
NSGA-II-CF	**0,77862 (0,02563)**	59,40 (5)	**0,67451 (0,00291)**	29,00 (29)	**0,75597 (0,17912)**	1,77 (2)	**0,25498 (0,00000)**	1,00 (1)
NSGA-II-MAB	0,74246 (0,05007)	57,80 (0)	**0,67264 (0,00470)**	29,00 (29)	**0,75437 (0,19636)**	1,77 (0)	**0,25498 (0,00000)**	1,00 (1)
SPEA2-CF	0,72687 (0,02653)	57,00 (0)	**0,67364 (0,00488)**	29,07 (28)	**0,69985 (0,19621)**	1,63 (0)	**0,25498 (0,00000)**	1,00 (1)
SPEA2-MAB	0,73881 (0,03911)	56,57 (0)	0,66782 (0,00917)	28,80 (29)	**0,60077 (0,22012)**	1,87 (2)	**0,25498 (0,00000)**	1,00 (1)
MOEAD-CF	**0,78657 (0,03590)**	38,80 (3)	0,66831 (0,00965)	29,13 (29)	**0,59486 (0,20803)**	1,57 (0)	**0,25498 (0,00000)**	1,00 (1)
MOEAD-MAB	**0,78928 (0,03579)**	37,80 (42)	0,67028 (0,00634)	29,17 (29)	**0,60982 (0,23393)**	1,67 (1)	**0,25498 (0,00000)**	1,00 (1)

Tabela 5.6: Resultados da comparação da HITO-DA com outros algoritmos para sistemas do contexto OA

Alg	**AJHsqldb**		**AJHotDraw**		**Health-Watcher**	
$Pf_{trueKnown}$	30		8		1	
	H_v	N_d	H_v	N_d	H_v	N_d
NSGA-II	0,60194 (0,09248)	30,23 (0)	0,62826 (0,09119)	4,10 (0)	**0,50495 (0,00000)**	1,00 (1)
SPEA2	0,57628 (0,08652)	28,13 (0)	0,54271 (0,11378)	4,73 (0)	**0,50495 (0,00000)**	1,00 (1)
NSGA-II-CF	0,69312 (0,10044)	32,40 (1)	**0,76708 (0,09000)**	5,17 (0)	**0,50495 (0,00000)**	1,00 (1)
NSGA-II-MAB	0,66027 (0,08791)	32,93 (2)	**0,71146 (0,16133)**	4,87 (0)	**0,50495 (0,00000)**	1,00 (1)
SPEA2-CF	0,60338 (0,05906)	28,27 (0)	**0,73146 (0,10801)**	4,33 (1)	**0,50495 (0,00000)**	1,00 (1)
SPEA2-MAB	0,61811 (0,08144)	25,07 (0)	**0,69768 (0,12865)**	5,40 (1)	**0,50495 (0,00000)**	1,00 (1)
MOEAD-CF	**0,73957 (0,07613)**	19,60 (22)	**0,72859 (0,08777)**	5,63 (0)	**0,50495 (0,00000)**	1,00 (1)
MOEAD-MAB	**0,78730 (0,06985)**	22,80 (5)	**0,73937 (0,13117)**	5,27 (7)	**0,48829 (0,09128)**	1,00 (1)

MOEAD-MAB em uma das 30 execuções encontrou uma solução diferente e dominada pela solução existente na fronteira $PF_{trueknown}$, desta forma obtendo uma média de *hypervolume* menor do que as outras instâncias.

5.5.2 Análise das fronteiras de Pareto

As figuras à seguir apresentam as Fronteiras de Pareto obtidas para cada sistema. Para todos os sistemas, todas as soluções das fronteiras PF_{known} foram apresentadas, mesmo que uma solução de um fronteira PF_{known} de outra instância a dominasse.

A Figura 5.1 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *MyBatis*. Nesta figura é possível visualizar que as fronteiras de Pareto geradas pelas instâncias MOEAD-MAB (com 42 soluções não dominadas), MOEAD-CF (com 3 soluções não dominadas) e NSGA-II-CF (com 5 soluções não dominadas) dominaram as soluções pertencentes a fronteira $PF_{trueknown}$.

A Figura 5.2 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *BCEL*. Nesta figura é possível visualizar que as instâncias de todos experimentos obtiveram a mesma fronteira PF_{known}. No entanto, nas instâncias NSGA-II, SPEA2, SPEA2-MAB, MOEAD-CF e MOEAD-MAB, a análise de *hypervolume* mostrou que estas instâncias obtiveram médias inferiores com diferença estatística. Isto ocorreu devido ao fato de que estas instâncias ao longo das 30 execuções encontraram soluções dominadas pelas instâncias de maior *hypervolume*.

A Figura 5.3 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *JHotDraw*. Nesta figura é possível visualizar que as fronteiras de Pareto geradas pelas instâncias NSGA-II-CF (com 2 soluções não dominadas), SPEA2-MAB (com 2 soluções não dominadas) e MOEAD-MAB (com 1 solução não dominada), dominaram as soluções pertencentes às fronteiras de Pareto de todas outras instâncias.

A Figura 5.4 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *JBoss*, onde todas as instâncias obtiveram a mesma fronteira PF_{known} contendo apenas uma solução.

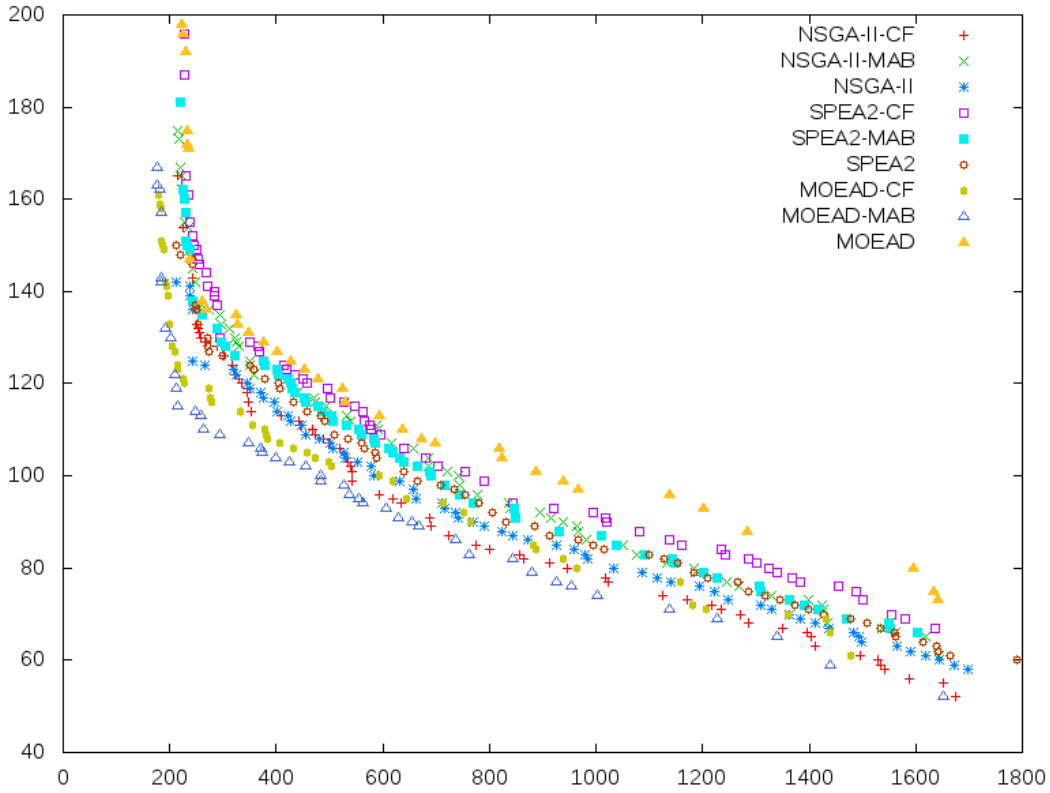


Figura 5.1: Fronteiras de Pareto para o sistema MyBatis

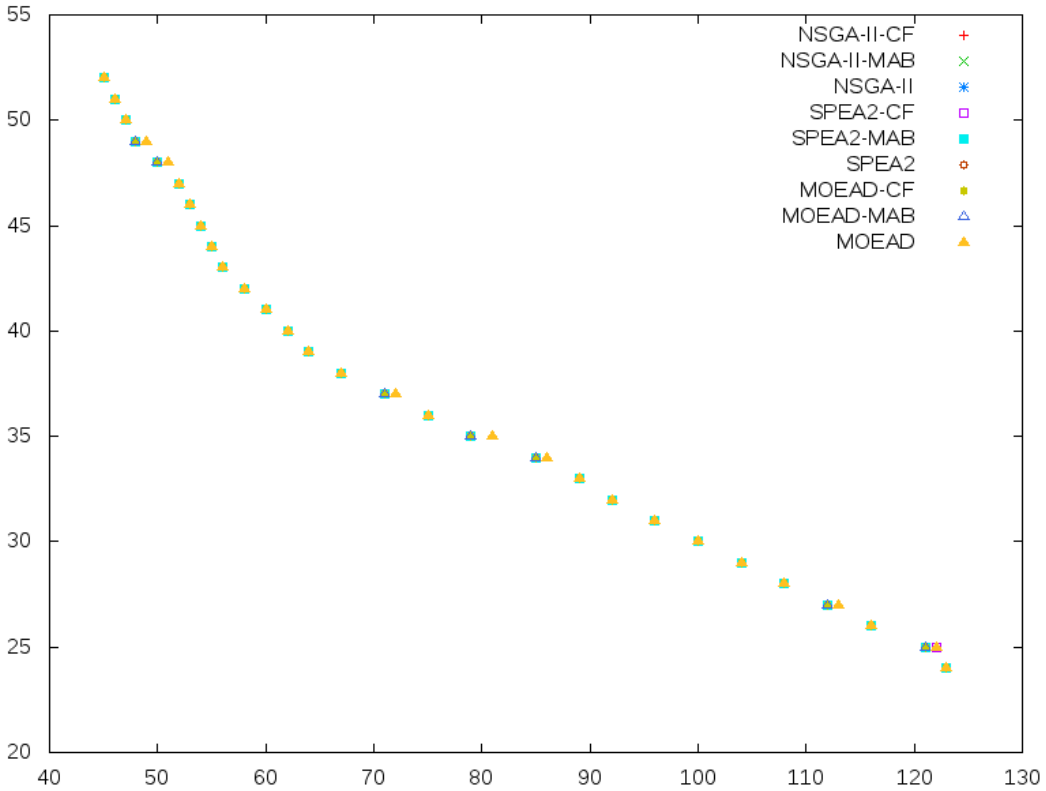


Figura 5.2: Fronteiras de Pareto para o sistema BCEL

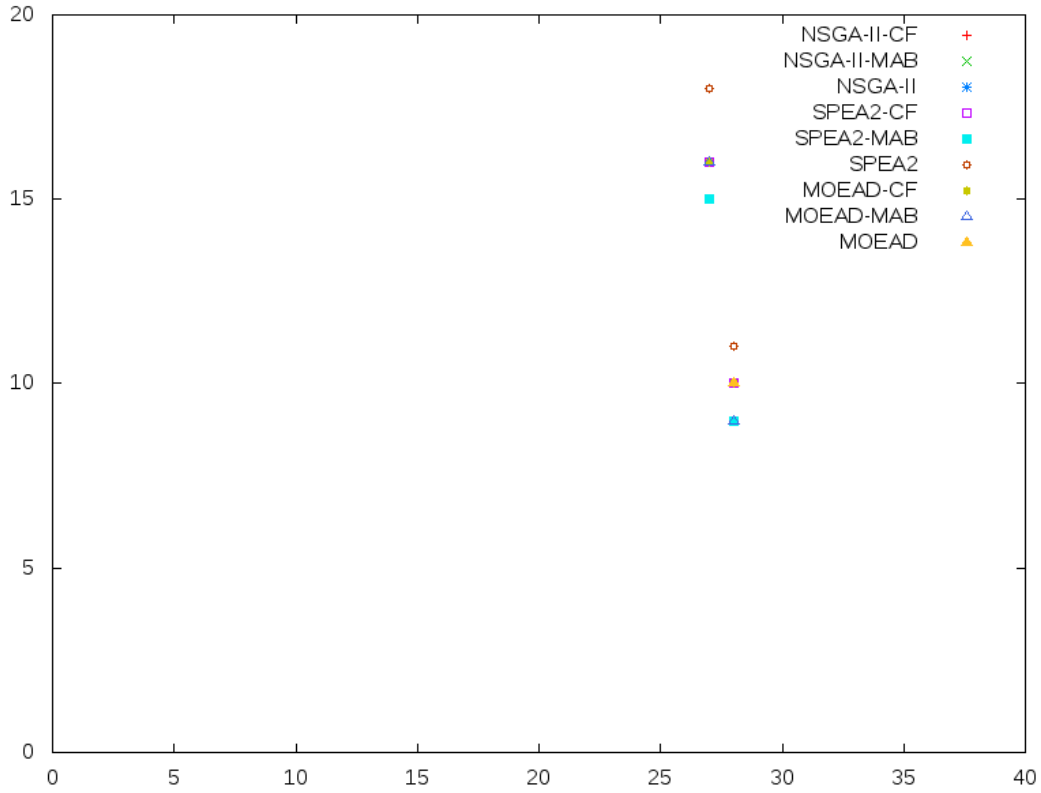


Figura 5.3: Fronteiras de Pareto para o sistema JHotDraw

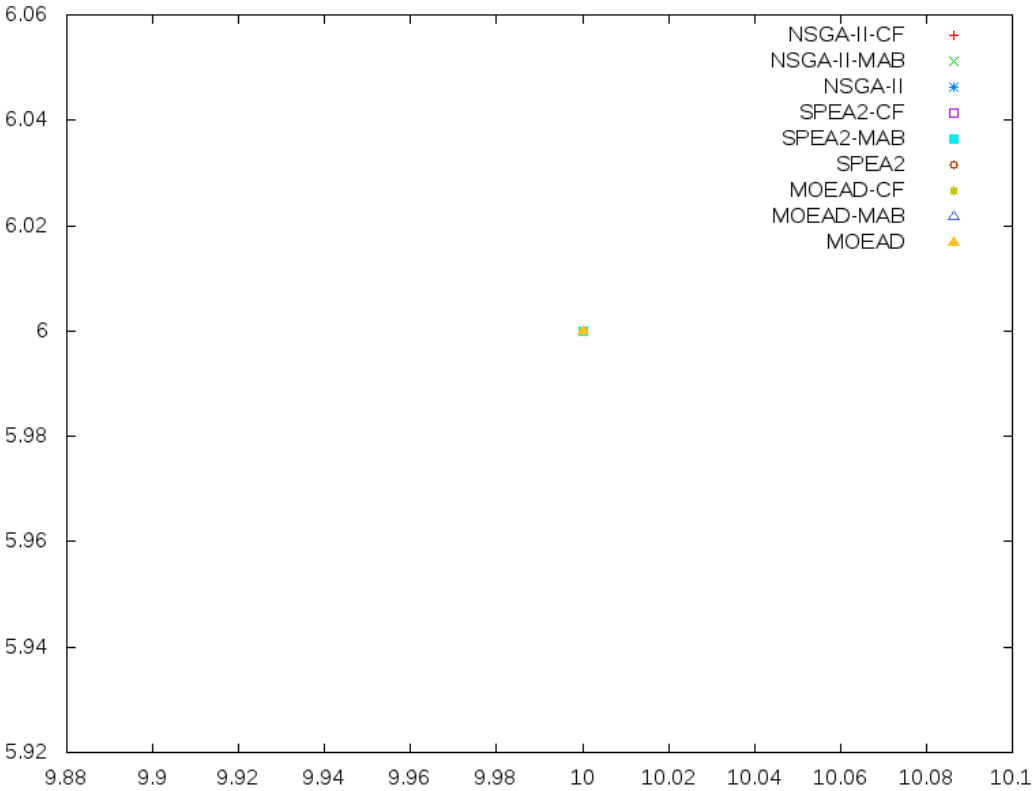


Figura 5.4: Fronteiras de Pareto para o sistema JBoss

A Figura 5.5 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *AJHsqldb*. Nesta figura é possível visualizar que as fronteiras de Pareto geradas pelas instâncias MOEAD-MAB (com 5 soluções não dominadas), MOEAD-CF (com 22 soluções não dominadas), NSGA-II-CF (com 1 solução não dominada) e NSGA-II-MAB (com 2 soluções não dominadas) dominaram as soluções pertencentes às fronteiras de Pareto de todas outras instâncias.

A Figura 5.6 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *AJHotDraw*. Nesta figura é possível visualizar que as fronteiras de Pareto geradas pelas instância MOEAD-CF (com 7 soluções não dominadas), SPEA2-MAB (com 1 solução não dominada) e SPEA2-CF (com 1 solução não dominada), dominaram as soluções pertencentes às fronteiras de Pareto de todas outras instâncias.

A Figura 5.7 apresenta as fronteiras de Pareto obtidas por todos os algoritmos para o sistema *Health-Watcher*, onde todas as instâncias obtiveram a mesma fronteira PF_{known} contendo apenas uma solução.

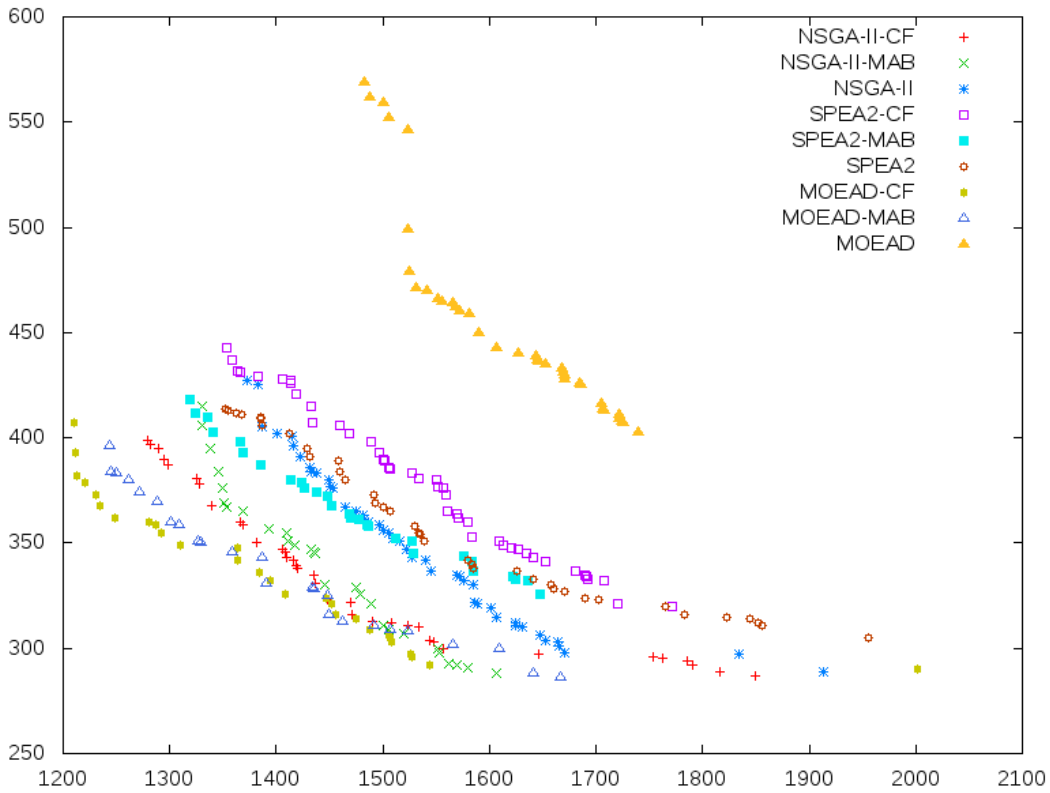


Figura 5.5: Fronteiras de Pareto para o sistema AJHsqldb

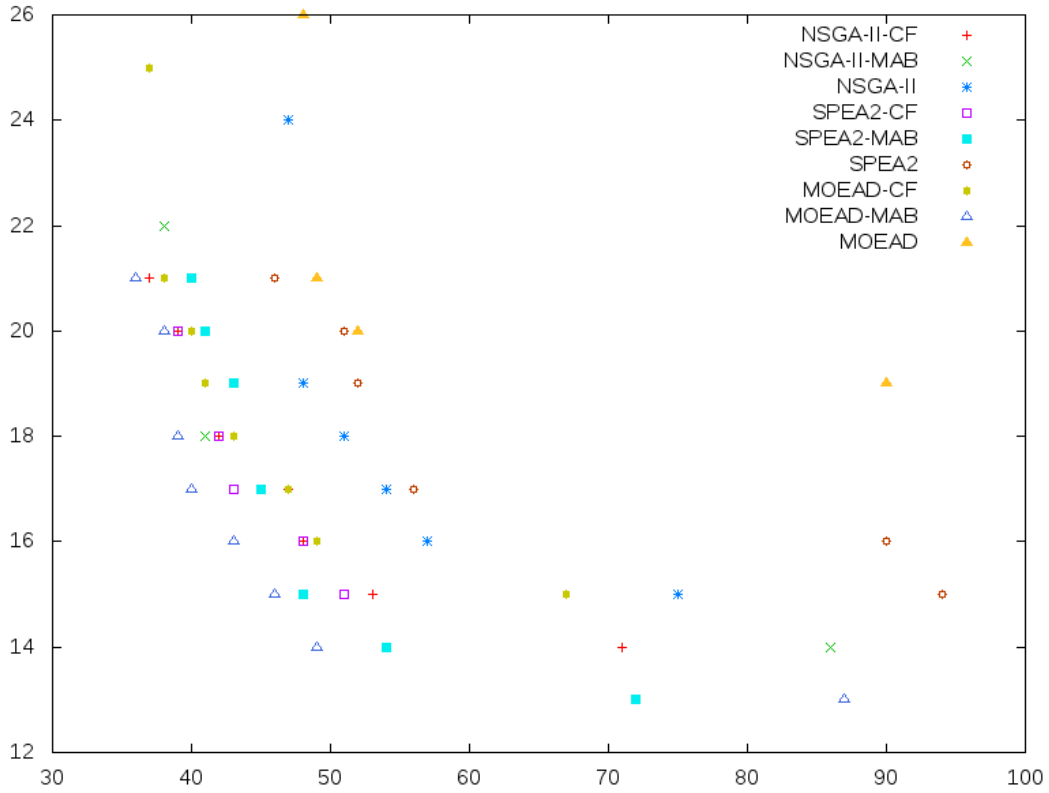


Figura 5.6: Fronteiras de Pareto para o sistema AJHotDraw

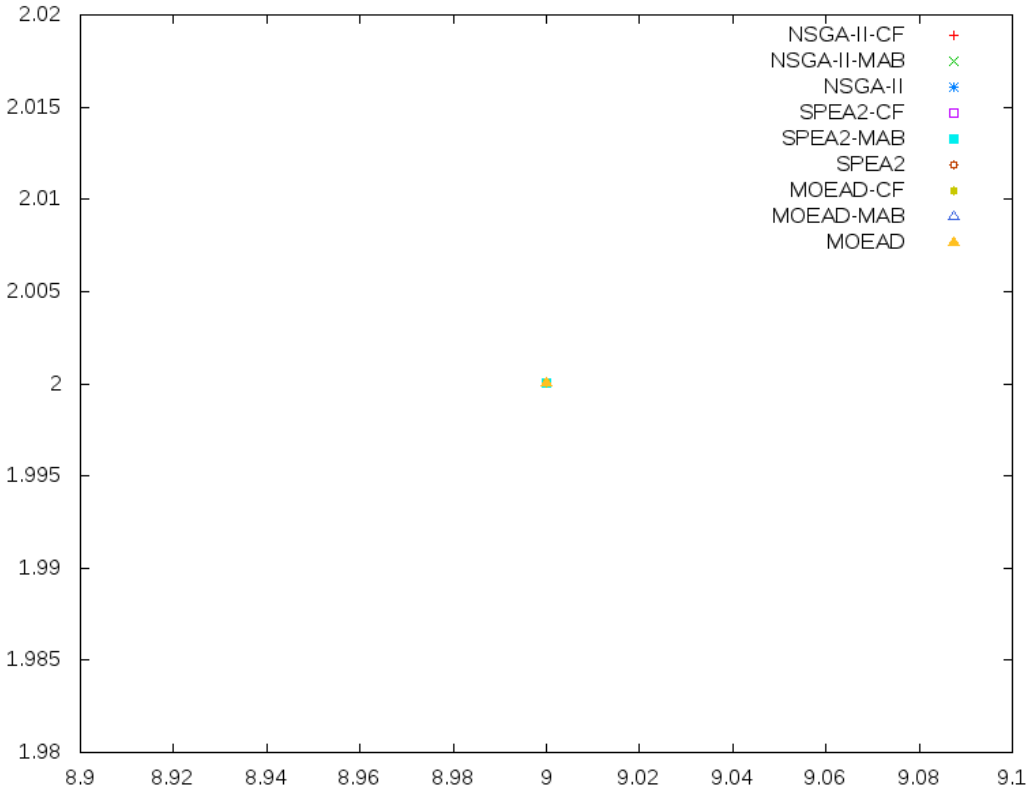


Figura 5.7: Fronteiras de Pareto para o sistema HealthWatcher

Dentre instâncias da HITO, todas instâncias obtiveram a mesma quantidade de soluções nas fronteiras PF_{known} nos sistemas *JBoss* e *Health-Watcher*. A instância NSGA-II-CF obteve fronteiras com maior quantidade de soluções nos sistemas *MyBatis*, *BCEL* (igual a instância NSGA-II-MAB) e *JHotDraw* (igual a instância SPEA2-MAB), obtendo assim um número maior de sistemas com fronteiras maiores. As instâncias NSGA-II-MAB, melhor no sistema *AJHsqldb* e empatada com a NSGA-II-CF no sistema *BCEL*, e a instância SPEA2-MAB, melhor no sistema *AJHotDraw* e empatada com a NSGA-II-CF no sistema *JHotDraw*, empataram em segundo lugar. A instância SPEA2-CF foi a pior das quatro instâncias nesta análise.

Ambas as instâncias da HITO-DA obtiveram a mesma quantidade de soluções nas fronteiras PF_{known} nos sistemas *JBoss*, *BCEL* e *Health-Watcher*. A instância MOEAD-MAB obteve fronteiras maiores nos sistemas *MyBatis*, *JHotDraw* e *AJHotDraw* do que a MOEAD-CF. A instância MOEAD-CF foi melhor apenas no sistema *AJHsqldb*. Assim constata-se uma maior quantidade de fronteiras maiores para a instância MOEAD-MAB.

Ao comparar todas as instâncias utilizadas (HITO, HITO-DA e meta-heurísticas), contata-se que todas instâncias obtiveram a mesma quantidade de soluções nas fronteiras PF_{known} nos sistemas *JBoss*, *BCEL* e *Health-Watcher*. A instância MOEAD-MAB obteve um número maior de soluções nas fronteiras *MyBatis* e *JHotDraw*, enquanto a instância MOEAD-CF obteve um maior número para o sistema *AJHsqldb* e a instância NSGA-II-CF para a instância *JHotDraw*. Desta forma, constata-se uma maior quantidade de fronteiras maiores para a instância MOEAD-MAB dentre todas instâncias utilizadas.

5.5.3 Análise das escolhas de heurísticas de baixo nível

A Tabela 5.7 mostra a média da quantidade de escolhas de cada LLH durante o processo da HITO-DA nas instâncias selecionadas. Devido ao fato das configurações das instâncias da HITO-DA usarem parâmetros menos elitistas, como $C = 5.0$ para o MOEAD-MAB e $\beta = 0.10$ para o MOEAD-CF, as escolhas da HITO-DA são bem distribuídas entre as heurísticas de baixo nível.

Em [28, 29] as escolhas da HITO foram muito mais elitistas do que as escolhas da HITO-DA, onde os operadores h1 *Two Points Crossover*, h4 *Uniform Crossover* e h7 *PMX Crossover* foram os mais escolhidos. Em alguns casos, o operador h1 foi empregado em pelo menos 32% dos escolhas.

Para a HITO-DA, configurações mais elitistas (com menores valores nos parâmetros C e β) escolheram mais um determinado operador em detrimento de outros, no entanto, com relação a *hypervolume*, seus resultados foram inferiores aos resultados obtidos pelas instâncias que selecionaram LLHs de forma uniforme.

Tabela 5.7: Média de escolhas de LLHs pela HITO-DA

Alg	Sistema	h1	h2	h3	h4	h5	h6	h7	h8	h9
MOEAD-CF	MyBatis	6599	6589	6592	6600	6621	6592	**6906**	6601	6595
	AJHsqldb	6578	6579	6578	6595	**6857**	6649	6616	6638	6604
	AJHotDraw	6620	6620	6619	6637	**6656**	6642	6625	6647	6631
	BCEL	6628	6633	6626	6640	6641	6641	6588	**6670**	6628
	JHotDraw	6607	6634	6633	6633	6646	6640	6597	**6658**	6647
	Health Watcher	6443	6662	6663	6649	6664	6671	6588	6673	**6684**
	JBoss	6632	6675	6672	6689	**6806**	6704	6157	6683	6677
MOEAD-MAB	MyBatis	**7483**	6695	7194	6393	6377	6426	6332	6343	6452
	AJHsqldb	**7518**	6686	7035	6338	6334	6338	6740	6340	6366
	AJHotDraw	**7870**	6445	6927	6334	6328	6329	6776	6336	6352
	BCEL	6848	6384	6547	6423	6331	6332	**8140**	6335	6356
	JHotDraw	**7621**	6477	6875	6346	6337	6338	7003	6347	6352
	Health Watcher	**7724**	6519	6937	6496	6333	6339	6637	6334	6376
	JBoss	7380	6405	6772	6333	6328	6331	**7475**	6333	6340

As Figuras 5.8 e 5.9 apresentam gráficos com as médias de escolha das LLHs ao longo das gerações. As médias foram obtidas ao longo de 30 execuções e cada geração possui 300 avaliações de *fitness*. Os resultados mostraram que em todos os sistemas a instância MOEAD-CF escolheu mais vezes uma mesma LLH durante poucas gerações. Já na instância MOEAD-MAB uma mesma LLH foi a mais escolhida durante uma quantidade maior de gerações, ainda que a média de escolhas fosse pouco maior.

As diferenças nas médias de escolhas entre as instâncias MOEAD-CF e MOEAD-MAB deve-se ao fato de que a FRRCF (usada pela MOEAD-CF) não usa o conceito de janela de tempo (empregado no FRRMAB) e desta forma dificulta que uma mesma LLH seja

a mais escolhida durante um período. Isto poderia ser resolvido diminuindo a influência do parâmetro β na equação da *Choice Function*, contudo experimentos mostraram que instâncias que usaram configurações mais elitistas não obtiveram bom desempenho com relação ao *hypervolume*.

5.5.4 Análise Estatística Geral

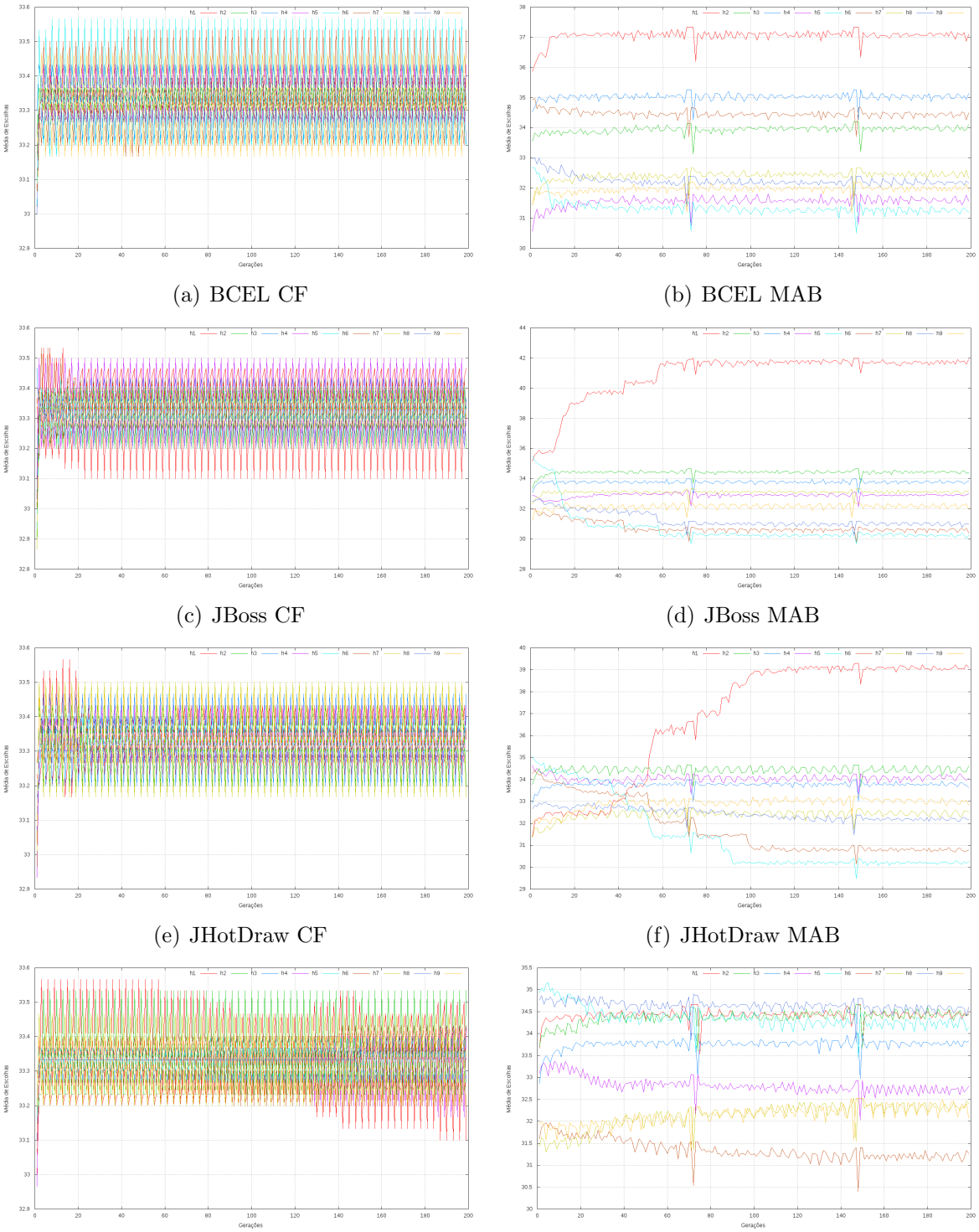
Devido aos bons resultados obtidos por instâncias da HITO e da HITO-DA as médias das classificações de Friedman foram calculadas a fim de verificar o desempenho geral das instâncias.

A Tabela 5.8 apresenta as médias das classificações de Friedman obtidas, baseadas no desempenho em todos sistemas da métrica *hypervolume*, onde quanto menor a média na classificação, melhor a instância é considerada.

A instância NSGA-II-CF da HITO obteve o melhor valor no teste em questão apesar do desempenho da instância MOEAD-MAB na análise de fronteiras de Pareto. A instância MOEAD-MAB foi a segunda melhor, superando as outras três instâncias da HITO (NSGA-II-MAB, SPEA2-CF e SPEA2-MAB) e a instância MOEAD-CF. A instância MOEAD-CF empatou com a instância SPEA2-CF na quarta posição e superou a instância SPEA2-MAB.

Tabela 5.8: Média de classificação das instâncias (Friedman)

Instâncias	Classificação
NSGA-II-CF	2.5
MOEAD-MAB	3.642857142857143
NSGA-II-MAB	3.9285714285714284
MOEAD-CF	4.214285714285714
SPEA2-CF	4.214285714285714
NSGA-II	5.357142857142857
SPEA2-MAB	5.642857142857142
SPEA2	6.499999999999999
MOEA/D	9.0



(a) BCEL CF

(b) BCEL MAB

(c) JBoss CF

(d) JBoss MAB

(e) JHotDraw CF

(f) JHotDraw MAB

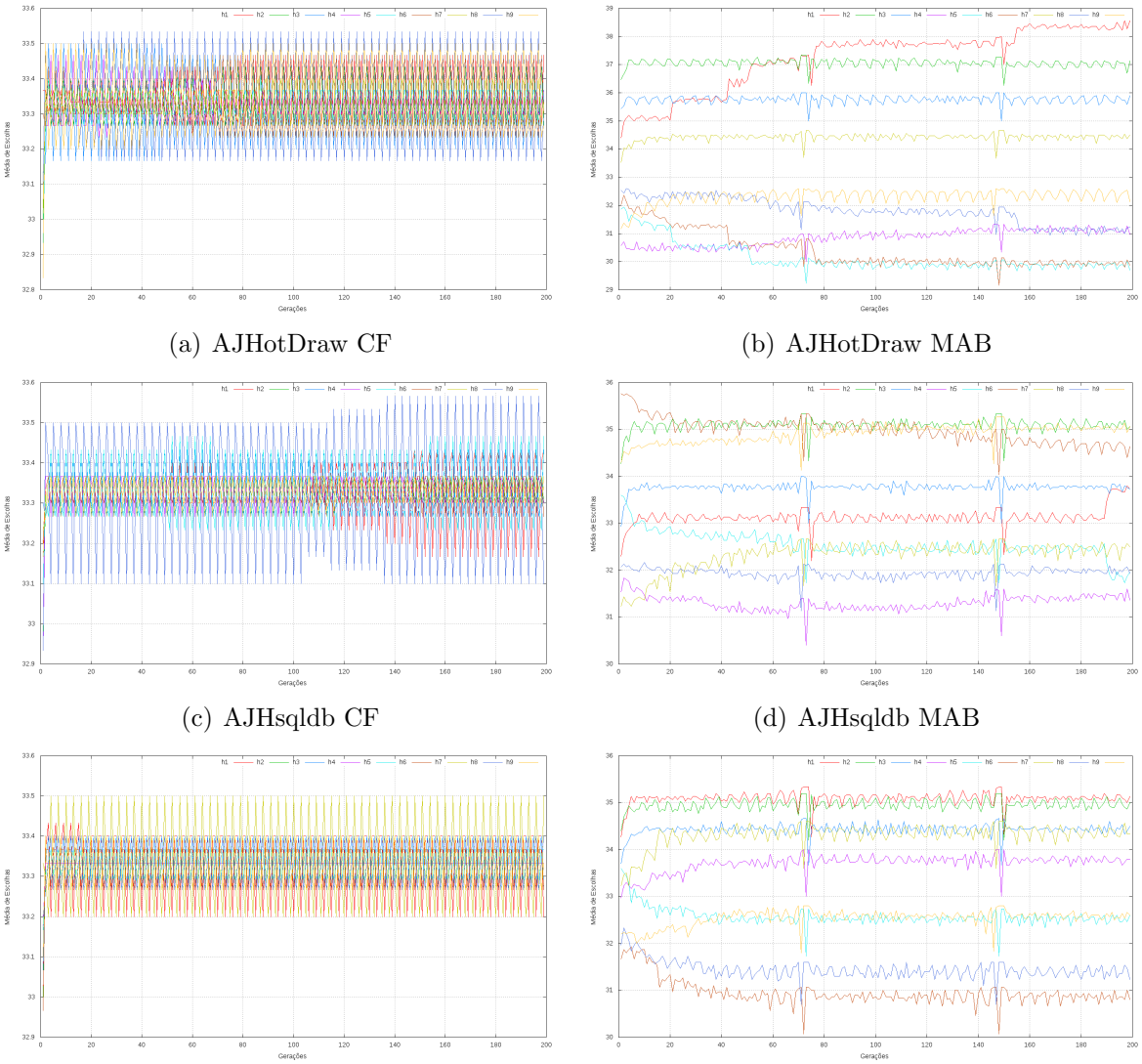
(g) MyBatis CF

(h) MyBatis MAB

Figura 5.8: Média de escolhas de LLHs para sistemas OO

5.5.5 Análise das soluções geradas

Nesta seção as soluções geradas são analisadas do ponto de vista do testador, de acordo como o valor de *fitness* e sequências obtidas. A Tabela 5.9 apresenta a análise das soluções



(a) AJHotDraw CF

(b) AJHotDraw MAB

(c) AJHsqldb CF

(d) AJHsqldb MAB

(e) HealthWatcher CF

(f) HealthWatcher MAB

Figura 5.9: Média de escolhas de LLHs para sistemas OA

geradas. Esta análise foi executada utilizando as soluções encontradas para o sistema $JHotDraw$, pois este sistema já foi selecionado para uma análise semelhante em [7]. Além disso, a quantidade de soluções não dominadas (pertencentes ao conjunto $PF_{trueknown}$) encontradas neste sistema permite este tipo de análise.

Na primeira coluna da Tabela 5.9 é mostrada a instância que obteve a solução, na segunda coluna os valores objetivos (Número de Atributos (A) e Número de Operações (O)) e na terceira coluna é apresentada a ordem de teste e integração.

Os resultados mostram que independentemente do algoritmo, as soluções podem ser consideradas boas para o testador, principalmente considerando a dificuldade de elas

serem geradas manualmente com tais valores ótimos de *fitness*. O testador deverá então escolher aquela que melhor se adeque à sua realidade, ambiente de desenvolvimento etc. Esta escolha poderá não estar somente baseada nos valores das funções objetivo. Pode-se observar que apesar das soluções possuírem valores semelhantes ou iguais, a sequência de classes dada para a integração difere. Caso a criação dos *Stubs* envolva a construção de operações complexas, então recomendam-se soluções com *fitness* igual a (28.0, 9.0). Caso a complexidade seja maior neste caso devido ao número levemente maior de atributos necessários, entao recomendam-se soluções com *fitness* igual a (27.0, 15.0).

Com relação à ordem gerada é possível constatar que as classes 87, 96, 24 sempre ocorrem no inicio das ordens, isto faz com que estas classes devam ser integradas e testadas primeiro.

5.6 Respondendo às questões de pesquisa

Respondendo à **RQ1**, tem-se que a HITO-DA apresentou melhores resultados, quando comparada à meta-heurística MOEA/D-DRA. Além disso, observa-se a vantagem de que o testador não necessita se preocupar com a escolha de um operador de cruzamento e mutação.

Respondendo à **RQ2**, observa-se que em quatro dos sete sistemas testados houve equivalências estatísticas com relação ao *hypervolume* entre todas instâncias da HITO e HITO-DA. Apenas para os sistemas *MyBatis*, *AJHsqldb* e *BCEL* observaram-se diferenças estatísticas entre as instâncias. No sistema *MyBatis* apenas a instância NSGA-II-CF foi equivalente às instâncias da HITO-DA. Na instância *BCEL* as instâncias NSGA-II-CF, NSGA-II-MAB e SPEA2-CF foram as que obtiveram as melhores médias de *hypervolume* e nenhuma instância da HITO-DA obteve resultados com equivalência estatística, ainda que as quantidades de soluções existentes na fronteira $PF_{trueknown}$ fossem iguais ao da HITO. A HITO-DA apresentou os melhores resultados para o sistema *AJHsqldb*, onde as instâncias da HITO-DA foram superiores a instâncias da HITO com relação ao *hypervolume* e quantidades de soluções na fronteira $PF_{trueknown}$.

Tabela 5.9: Soluções não dominadas para o sistema JHotDraw

Instância	Obj. (A, O)	Ordem
MOEAD-MAB	28.0, 9.0	87 96 186 114 157 24 84 191 147 190 82 99 115 101 121 47 23 188 53 180 25 105 102 116 117 138 17 49 133 97 14 56 12 51 15 77 9 154 58 36 85 106 155 20 127 0 7 111 128 140 184 86 118 22 142 136 16 156 187 37 10 189 6 42 34 108 11 173 146 19 3 61 28 26 76 153 107 192 143 145 104 21 131 163 185 30 48 27 158 182 63 33 5 119 152 91 183 32 31 160 78 73 66 18 54 139 161 109 52 79 135 50 132 55 62 45 172 41 75 178 46 92 39 123 83 81 148 103 164 166 124 159 168 68 196 70 125 144 71 93 64 167 59 72 69 170 130 90 149 175 29 80 134 171 94 89 195 162 13 43 35 4 165 1 60 95 141 98 181 67 100 194 113 57 112 174 120 88 110 137 8 2 193 38 179 126 150 44 176 122 40 65 151 177 169 129 74
NSGAII-CF	28.0, 9.0	87 96 24 84 114 102 191 17 138 15 47 186 99 0 123 48 79 116 77 121 157 9 25 58 190 7 180 111 188 133 10 89 164 49 22 104 154 20 56 93 143 91 163 101 115 85 184 147 142 128 100 117 12 131 82 118 105 53 23 195 19 16 149 160 127 153 36 119 155 187 106 6 124 156 50 18 168 42 183 11 69 146 97 196 34 189 181 103 165 88 3 81 28 108 67 68 46 158 51 29 173 30 76 32 37 66 57 182 27 31 185 75 62 161 2 4 136 90 175 113 14 152 166 135 83 71 112 33 192 5 98 73 64 13 141 86 74 70 54 178 55 39 140 26 139 35 159 59 174 126 44 172 63 41 92 167 52 61 132 40 193 21 134 194 148 145 120 109 125 95 170 78 43 144 45 107 177 80 130 1 60 8 65 94 169 162 110 72 38 171 150 151 137 179 176 122 129
NSGAII-CF	27.0, 15.0	87 24 96 84 114 102 191 17 138 15 47 186 99 0 123 48 79 116 77 147 157 25 9 58 101 190 7 180 115 111 188 133 10 128 89 164 49 154 104 56 22 20 100 85 19 143 91 184 117 163 121 105 93 195 82 118 142 53 23 131 12 16 149 160 127 153 36 119 155 187 106 6 124 156 50 18 168 42 183 11 69 146 97 196 34 189 181 103 165 88 3 81 28 108 67 68 46 158 51 29 173 30 76 32 37 66 57 182 27 31 185 75 62 161 2 4 136 90 175 113 14 152 166 135 83 71 112 192 5 98 73 64 13 141 86 74 70 54 178 55 26 35 159 59 174 126 44 172 63 41 92 167 52 61 132 21 148 140 40 33 193 95 107 177 94 144 134 130 125 139 170 45 145 194 120 109 78 43 80 1 60 39 8 65 171 162 72 110 150 169 179 176 122 38 137 151 129
SPEA2-MAB	28.0, 9.0	87 96 24 190 25 84 46 180 191 115 47 56 49 114 173 10 102 157 99 188 101 17 77 93 53 133 58 184 100 104 164 123 138 82 116 85 186 22 128 76 15 68 127 0 79 97 78 147 117 103 81 9 196 6 12 142 50 14 26 106 154 195 160 118 131 161 19 183 63 187 89 18 20 105 36 42 48 23 74 163 69 155 181 11 3 51 21 34 16 156 153 189 7 143 28 67 13 168 146 149 119 136 192 165 4 113 112 92 111 62 108 29 75 64 88 109 30 121 37 71 170 194 98 158 152 57 125 185 91 86 124 73 150 193 41 66 95 61 140 182 80 27 159 70 120 59 55 33 35 83 148 178 175 31 167 72 166 139 162 5 177 32 90 2 54 40 43 39 172 52 107 135 45 132 144 134 141 145 38 122 1 126 174 44 171 94 60 8 110 137 65 151 179 176 130 169 129
SPEA2-MAB	27.0, 15.0	87 24 96 191 102 138 84 56 99 116 186 115 17 47 49 53 114 101 25 100 77 15 188 85 10 157 142 58 180 117 154 48 184 149 128 195 19 50 12 23 173 161 20 143 6 26 42 9 163 36 190 7 0 93 147 62 63 127 69 82 155 11 97 98 187 164 18 131 81 16 156 3 119 76 34 158 189 183 111 28 141 139 91 133 30 168 108 118 22 105 121 159 79 125 170 103 104 67 46 86 33 29 51 196 78 152 123 13 146 140 45 92 27 89 162 106 182 59 178 4 66 144 73 32 136 1 192 90 110 113 75 150 37 8 65 194 14 160 21 166 175 61 31 5 52 124 145 54 135 185 169 107 132 130 153 83 148 57 109 137 68 2 88 174 55 74 80 120 165 112 151 64 35 181 134 177 126 70 71 167 72 172 39 41 60 40 43 95 193 38 44 179 129 94 171 122 176

Ao analisar as fronteiras PF_{known} pode-se verificar nos sistemas *BCEL*, *JBoss* e *HealthWatcher* que todas instâncias obtiveram a mesma quantidade de soluções não dominadas. No sistema *MyBatis* as instâncias da HITO-DA obtiveram uma quantidade maior de soluções não dominadas na fronteira $PF_{trueknown}$ (MOEAD-MAB (42), NSGA-II-CF (5) e MOEAD-CF (3)). No sistema *AJHsqldb* o mesmo ocorreu, e instâncias da HITO-DA obtiveram melhores resultados (MOEAD-CF (22), MOEAD-CF (5), NSGA-II-MAB (2) e NSGA-II-CF (1)). No sistema *JHotDraw* a instância NSGA-II-CF e SPEA2-MAB possuem duas soluções na fronteira $PF_{trueknown}$, uma quantidade maior do que encontrada em instâncias da HITO-DA. No sistema *AJHotDraw* a instância MOEAD-MAB da HITO-DA obteve uma quantidade maior de soluções em relação à HITO (MOEAD-MAB (7), SPEA2-CF (1) e SPEA2-MAB (1)). Assim pode-se concluir que, geralmente, instâncias da HITO-DA possuem melhor desempenho com relação à fronteira PF_{known} gerada.

Os resultados da média de classificação de Friedman apontaram a instância NSGA-II-CF da HITO como a melhor instância em termos gerais com relação a métrica *hypervolume*. No entanto, observa-se um melhor desempenho da hiper-heurística HITO-DA em sistemas maiores em relação à métrica *hypervolume*, e melhores resultados com relação à análise das fronteiras PF_{known}, onde a HITO-DA obteve as melhores fronteiras nos maiores sistemas e fronteiras equivalentes em sistemas menores.

5.7 Ameaças à Validade

A principal ameaça à validade deste trabalho é que alguns sistemas são pequenos. Para estes sistemas os algoritmos usados não apresentam diferenças em seus resultados, pois todos são capazes de encontrar a solução ótima. Outros experimentos com sistemas maiores devem ser conduzidos no futuro.

Outra ameaça são os valores utilizados para o os parâmetros dos algoritmos. Para minimizar isto, procurou-se utilizar valores da literatura e testar diferentes combinações, procurando encontrar a melhor configuração de parâmetros para cada instância.

5.8 Considerações Finais

Neste capítulo os experimentos foram descritos, e os resultados apresentados e avaliados. Os experimentos foram realizados utilizando as hiper-heurísticas HITO-DA e HITO, e as meta-heurísticas NSGA-II, SPEA2 e MOEA/D, e sete diferentes sistemas. A análise dos resultados foi realizada com base nos valores de *hypervolume* e quantidade de soluções não dominadas. Os resultados obtidos pela HITO-DA foram melhores em sistemas maiores como o *MyBatis*, e principalmente o sistema *AJHsqldb* enquanto instâncias da HITO obtiveram melhores resultados em sistemas menores como o (*BCEL*).

CAPÍTULO 6

CONCLUSÕES

Este trabalho introduziu a hiper-heurística HITO-DA (*Hyper-heuristic for the Integration and Test Order Problem using Decomposition Approach*), que é uma hiper-heurística que incorpora algumas características da hiper-heurística HITO (*Hyper-heuristic for the Integration and Test Order Problem*) e trabalha com algoritmos de decomposição, a fim de encontrar soluções para o problema de estabelecer uma sequência de módulos para o teste de integração (*Integration Test Order Problem* (ITO)).

A HITO-DA foi instanciada usando a meta-heurística MOEA/D-DRA e os algoritmos de seleção FRRMAB (*Fitness Rate Rank Multi Armed Bandit*) e FRRCF (*Fitness Rate Rank with Choice Function*). O FRRCF foi proposto no presente trabalho combinando características do FRRMAB e CF (*Choice Function*). Isto foi motivado devido aos resultados obtidos pela HITO em suas instâncias que usaram o algoritmo de seleção CF.

Experimentos foram conduzidos para verificar o comportamento da HITO-DA em relação às meta-heurísticas e instâncias da HITO. Os experimentos foram realizados para sete sistemas reais, e os resultados foram analisados e comparados estatisticamente.

Os resultados do estudo empírico conduzido mostraram que a MOEAD-MAB obteve melhores resultados do que a MOEAD-CF. No entanto os bons resultados da MOEAD-CF no sistema *AJHsqldb* ainda motivam sua aplicação em sistemas maiores e mais complexos. Ao comparar a hiper-heurística proposta com a meta-heurística MOEA/D-DRA, observou-se que a HITO-DA obteve melhores resultados do que a meta-heurística MOEA/D em todos os casos. Ao comparar a hiper-heurística proposta com a HITO, constatou-se que a HITO-DA possui melhor desempenho em sistemas maiores, e que a HITO possui um melhor desempenho geral quando considerados sistemas menores. Portanto, recomenda-se utilizar a HITO-DA em sistemas maiores e complexos, e utilizar a HITO em casos gerais.

6.1 Trabalhos Futuros

Para dar continuidade ao presente trabalho, alguns trabalhos futuros foram identificados. Dentre eles:

- Realização de experimentos com sistemas maiores e mais complexos, casos nos quais a aplicação da HITO-DA parece ser mais interessante;
- Instanciar a HITO-DA com outros algoritmos de seleção, como o AdaptHH [43];
- Instanciar a HITO-DA com outras meta-heurísticas baseadas em decomposição, tais como o MOEA/D-DD [39] e MOEA/D-AWA [46];
- Estudar o comportamento da HITO-DA para se resolver o problema ITO com um número maior de objetivos;
- Adaptar a HITO-DA para trabalhar com o problema ITO na presença de restrições de modularização que definem que certos grupos de módulos precisam ser desenvolvidos e testados em conjunto;
- Criar uma variação da HITO-DA para outros problemas da SBSE, tais como a seleção ou priorização de casos de teste;

REFERÊNCIAS

[1] S. F. Adra. *Improving Convergence, Diversity and Pertinency in Multiobjective Optimisation*. Tese de Doutorado, Department of Automatic Control and Systems Engineering, The University of Sheffield, 2007.

[2] W. K. G. Assunção. Uma abordagem para integração e teste de módulos baseada em agrupamentos e algoritmos de otimização multiobjetivos. Dissertação de Mestrado, UFPR, 2012.

[3] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. T. R. Pozo. Estabelecendo sequências de teste de integraçao de classes: Um estudo comparativo da aplicação de três algoritmos evolutivos multiobjetivos. *Workshop de Teste e Tolerância a Falhas*, 2011.

[4] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. T. R. Pozo. Generating integration test orders for aspect oriented software with multi-objective algorithms. *Revista de Informática Teórica e Aplicada*, páginas 301–327, 2013.

[5] W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, e S. R. Vergilio. Establishing integration test orders of classes with several coupling measures. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO, páginas 1867–1874, New York, NY, USA, 2011. ACM.

[6] W. K. G. Assunção, T. E. Colanzi, A. T. R. Pozo, e S. R. Vergilio. Uma avaliação do uso de diferentes algoritmos evolutivos multiobjetivos para integração de classes e aspectos. *II Workshop on Search Based Software Engineering*, 2011.

[7] W. K. G. Assunção, T. E. Colanzi, S. R. Vergilio, e A. Pozo. A multi-objective optimization approach for the integration and test order problem. *Information Science*, páginas 119–139, Maio de 2014.

[8] P. Auer, N. Cesa-Bianchi, e P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, páginas 235–256, Maio de 2002.

[9] M. P. Basgalupp, R. C. Barros, T. S. da Silva, e A. C. P. L. F. de Carvalho. Software effort prediction: A hyper-heuristic decision-tree based approach. *Proceedings of the 28th ACM Symposium on Applied Computing*, páginas 1109–1116, 2013.

[10] V. Bowman. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. *Multiple Criteria Decision Making*, volume 130 of *Lecture Notes in Economics and Mathematical Systems*, páginas 76–86. Springer Berlin Heidelberg, 1976.

[11] L. C. Briand, J. Feng, e Y. Labiche. Using genetic algorithms and coupling measures to devise optimal integration test orders. *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE 02, páginas 43–50, New York, NY, USA, 2002. ACM.

[12] L. C. Briand, J. Feng, e Y. Labiche. Experimenting with genetic algorithms to devise optimal integration test orders. T. M. Khoshgoftaar, editor, *Software Engineering with Computational Intelligence*, volume 731 of *The Springer International Series in Engineering and Computer Science*, páginas 204–234. Springer US, 2003.

[13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, e R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, páginas 1695–1724, Dezembro de 2013.

[14] R. da V. Cabral, A. Pozo, e S. R. Vergilio. A Pareto ant colony algorithm applied to the class integration and test order problem. A. Petrenko, Adenilso S., e J. C. Maldonado, editors, *Testing Software and Systems*, volume 6435 of *Lecture Notes in Computer Science*, páginas 16–29. Springer Berlin Heidelberg, 2010.

[15] V. R. Carvalho, S. R. Vergilio, e A. T. R. Pozo. Uma hiper-heurística de seleção de meta-heurísticas para estabelecer sequências de módulos para o teste de software. *Workshop de Engenharia de Software Baseada em Busca*, 2015.

[16] C. A. C. Coello. *Evolutionary algorithms for solving multi-objective problems*. Springer, New York, 2007.

[17] T. E. Colanzi. *Uma abordagem de otimização multiobjetivo para projeto arquitetural de linha de produto de software*. Tese de Doutorado, Univesidade Federal do Paraná (UFPR), Brasil, 2014.

[18] P. I. Cowling, G. Kendall, e E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling*, páginas 176–190, 2001.

[19] K. Deb, A. Pratap, S. Agarwal, e T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, páginas 182–197, Abril de 2002.

[20] J. Derrac, S. Garcia, D. Molina, e F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, páginas 3–18, 2011.

[21] J. Derrac, S. García, D. Molina, e F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, páginas 3 – 18, 2011.

[22] M. Dorigo e G. Di Caro. Ant colony optimization: a new meta-heuristic. *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, 1999.

[23] J. Durillo e A. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, páginas 760–771, 2011.

[24] Á. Fialho. *Adaptive Operator Selection for Optimization*. Tese de Doutorado, École Doctorale d' Informatique, 2010.

[25] M. Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, páginas 675–701, Dezembro de 1937.

[26] R. Galvan, A. T. R. Pozo, e S. R. Vergilio. Establishing integration test orders for aspect-oriented programs with an evolutionary strategy. *Latinamerican Workshop on Aspect Oriented Software*, 2010.

[27] D. E. Goldberg e J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, páginas 95–99, 1988.

[28] G. Guizzo, G. M. Fritsche, S. R. Vergilio, e A. T. R. Pozo. A hyper-heuristic for the multi-objective integration and test order problem. *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, GECCO, páginas 1343–1350, New York, NY, USA, 2015. ACM.

[29] G. Guizzo, S. R. Vergilio, e A. T. R. Pozo. Evaluating a multi-objective hyper-heuristic for the integration and test order problem. *The 4th Brazilian Conference on Intelligent Systems (BRACIS)*, 2015.

[30] M. Harman, E. Burke, J. Clark, e X. Yao. Dynamic adaptive search based software engineering. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, páginas 1–8, 2012.

[31] M. Harman, S. A. Mansouri, e Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, páginas 11:1–11:61, Dezembro de 2012.

[32] Y. Jia, M. Cohen, M. Harman, e J. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, Maio de 2015.

[33] L. P. Kaelbling, M. L. Littman, e A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, páginas 237–285, 1996.

[34] J. Kennedy e R. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks, 1995.*, volume 4, páginas 1942–1948 vol.4, Novembro de 1995.

[35] J. Knowles, L. Thiele, e E. Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. Relatório técnico, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, 2006.

[36] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[37] A.C. Kumari, K. Srinivas, e M.P. Gupta. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. *2013 IEEE 3rd International Advance Computing Conference (IACC)*, páginas 813–818, Fevereiro de 2013.

[38] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, e C. Chen. A test strategy for object-oriented programs. *Proceedings of Nineteenth Annual International Computer Software and Applications Conference (COMPSAC 95), 1995*, páginas 239–244, Agosto de 1995.

[39] K. Li, K. Deb, Q. Zhang, e S. Kwong. An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Transactions on Evolutionary Computation*, páginas 694–716, Outubro de 2015.

[40] K. Li, A. Fialho, S. Kwong, e Q. Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, Fevereiro de 2014.

[41] M. Maashi, E. Özcan, e G. Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications*, páginas 4475–4493, 2014.

[42] H. Melton e E. Tempero. An empirical study of cycles among classes in Java. *Empirical Software Engineering*, páginas 389–415, Agosto de 2007.

[43] M. Misir, K. Verbeeck, P. De Causmaecker, e G. Vanden Berghe. A new hyper-heuristic implementation in hyflex: a study on generality. *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory and Application*, páginas 374–393, 2011.

[44] V. Pareto. *Manuel D Economie Politique*. Ams Press, Paris, 1927.

[45] K. Praditwong, M. Harman, e X. Yao. Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*, páginas 264–282, Março de 2011.

[46] Y. Qi, X. Ma, F. Liu, L. Jiao, J. Sun, e J. Wu. MOEA/D with adaptive weight adjustment. *Evolutionary Computation*, páginas 231–264, Junho de 2014.

[47] R. Ré. *Uma contribuição para a minimização do numero de stubs no teste de integração de programas orientados a aspectos*. Tese de Doutorado, ICMC/USP, 2009.

[48] R. Ré, O. Lemos, A. Lazzarini, e P. C. Masiero. Minimizing stub creation during integration test of aspect-oriented programs. *Proceedings of the 3rd Workshop on Testing Aspect-oriented Programs*, WTAOP 07, páginas 1–6, New York, NY, USA, 2007. ACM.

[49] R. Ré e P. C. Masiero. Integration testing of aspect-oriented programs: a characterization study to evaluate how to minimize the number of stubs. *Brazilian Symposium on Software Engineering*, páginas 411–426, 2007.

[50] N.R. Sabar, M. Ayob, G. Kendall, e R. Qu. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, páginas 217–228, Fevereiro de 2015.

[51] S. R. Vergilio, A. T. R. Pozo, J. C. G. Árias, R. da V. Cabral, e T. Nobre. Multi-objective optimization algorithms applied to the class integration and test order problem. *International Journal on Software Tools for Technology Transfer*, páginas 461–475, 2012.

[52] Z. Wang, Q. Zhang, A. Zhou, M. Gong, e L. Jiao. Adaptive replacement strategies for moea/d. *IEEE Transactions on Cybernetics*, PP(99), 2015.

[53] Q. Zhang e H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, Dezembro de 2007.

[54] Q. Zhang, W. Liu, e H. Li. The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. Relatório Técnico CES-491, School of CS & EE, University of Essex, Fevereiro de 2009.

[55] Q. Zhang e P. N. Suganthan. Final report on cec'09 moea competition. *Congress on evolutionary computation (CEC 2009)*, 2009.

[56] E. Zitzler e S. Künzli. Indicator-based selection in multiobjective search. *PPSN*, volume 3242 of *Lecture Notes in Computer Science*, páginas 832–842. Springer, 2004.

[57] E. Zitzler, M. Laumanns, e L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, páginas 95–100. International Center for Numerical Methods in Engineering, 2001.

[58] E. Zitzler e L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, Novembro de 1999.

APÊNDICE A

RESULTADOS DO TESTE DE PARÂMETROS

Este apêndice apresenta os resultados do teste de parâmetros detalhado no Capítulo 5. Todas as configurações testadas para a HITO-DA utilizaram os parâmetros $\delta = 0.9$ e $T = 30$, e todas as configurações do FRRCF usaram o parâmetro $\alpha = 1.0$. Em todas as tabelas os valores em **negrito** representam equivalência estatística segundo o teste estatístico de Kruskal-Wallis, e na coluna *Rank*, a média de ranking obtida no teste estatístico de Friedman. Os valores selecionados são valores que possuem a maior quantidade de equivalências estatísticas, que neste teste de parâmetros foi 7 em todos os casos, e menor valor de *Rank* segundo o teste de Friedman. As configurações que atendam estes quesitos estão destacadas.

As Tabelas A.1 e A.2 apresentam o teste de parâmetros para a HITO-DA. A Tabela A.7 apresenta o resultado do teste de parâmetros para o MOEA/D. Por fim, as Tabelas A.3, A.4, A.5 e A.6 apresentam os resultados para o teste de parâmetros da HITO.

Tabela A.1: Resultados de *tuning* para configurações do FRRMAB

Id	Rank	**Nr**	**D**	**C**	**W**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
1	120.43	30	0.7	0.5	300	**0,69799 (0,10659)**	0,59925 (0,19051)	**0,66348 (0,18883)**	0,74361 (0,03197)	0,62040 (0,17929)	**0,98755 (0,03632)**	**0,95793 (0,10896)**
2	107.64	30	0.9	0.5	300	0,66645 (0,09329)	**0,64800 (0,13310)**	**0,60466 (0,17055)**	**0,75600 (0,01461)**	0,64313 (0,19183)	**0,99137 (0,02801)**	**0,99253 (0,04091)**
3	125.57	30	0.5	0.5	300	0,67299 (0,10473)	**0,64761 (0,15344)**	**0,60194 (0,17121)**	0,74081 (0,03322)	0,57267 (0,18901)	**0,96842 (0,11132)**	**0,95557 (0,10879)**
4	106.14	6	1.0	0.5	300	0,68842 (0,07104)	0,55517 (0,11312)	**0,61913 (0,17507)**	**0,75173 (0,02429)**	**0,69911 (0,18740)**	**0,99257 (0,02845)**	**0,99293 (0,02158)**
5	102.43	3	0.1	0.5	300	0,68396 (0,07754)	0,44705 (0,14778)	**0,62521 (0,17076)**	**0,75245 (0,02395)**	**0,78887 (0,16022)**	**0,99273 (0,03113)**	**0,99764 (0,01291)**
6	91.64	3	0.3	0.5	300	0,66261 (0,08396)	0,44641 (0,15900)	0,60212 (0,15471)	**0,75362 (0,02157)**	**0,78782 (0,13025)**	**1,00000 (0,00000)**	**0,99529 (0,01794)**
7	118.71	30	0.3	0.5	300	**0,70187 (0,10664)**	**0,65560 (0,16052)**	0,56604 (0,18683)	0,74766 (0,02416)	0,63827 (0,13600)	**0,99152 (0,02341)**	**0,93121 (0,14047)**
8	87.21	3	0.7	0.5	300	0,63704 (0,09156)	0,45368 (0,14944)	**0,64844 (0,15655)**	**0,75686 (0,02080)**	**0,76108 (0,15318)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
9	124.43	30	0.1	0.5	300	0,68679 (0,08507)	0,61611 (0,15116)	0,58285 (0,19560)	**0,75506 (0,02387)**	0,58532 (0,15154)	**0,98770 (0,03065)**	**0,93616 (0,18606)**
10	92.71	3	0.5	0.5	300	0,64761 (0,08387)	0,42530 (0,12087)	**0,61789 (0,17021)**	**0,75750 (0,01193)**	**0,77875 (0,15414)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
11	103.50	3	0.9	0.5	300	0,65670 (0,08099)	0,46356 (0,13275)	**0,66140 (0,17377)**	**0,75215 (0,02098)**	**0,73619 (0,15086)**	**0,99917 (0,00452)**	**0,99017 (0,04247)**
12	93.79	3	1.0	0.5	150	0,64877 (0,07654)	0,46798 (0,09851)	**0,63053 (0,14150)**	**0,75708 (0,01568)**	**0,79473 (0,15170)**	**1,00000 (0,00000)**	**0,98310 (0,04611)**
13	105.29	30	1.0	0.5	150	**0,72028 (0,08067)**	**0,68882 (0,10825)**	0,61193 (0,13457)	**0,75621 (0,01289)**	0,62061 (0,11267)	**0,97848 (0,06777)**	**0,97368 (0,08696)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
14	113.71	30	0.9	0.5	150	**0,70994 (0,06791)**	**0,62601 (0,15164)**	**0,63286 (0,14221)**	0,75603 (0,01036)	0,62216 (0,15511)	**0,98331 (0,06093)**	**0,95438 (0,12617)**
15	107.43	30	0.5	0.5	150	**0,70435 (0,08620)**	0,62916 (0,17817)	**0,66088 (0,16123)**	0,74651 (0,03000)	**0,67205 (0,14481)**	**0,99752 (0,00996)**	**0,96618 (0,11552)**
16	108.71	6	0.1	0.5	300	**0,69349 (0,11712)**	0,48482 (0,15843)	**0,64477 (0,16649)**	**0,75078 (0,02207)**	**0,75947 (0,17211)**	**0,99752 (0,00755)**	**0,97523 (0,08579)**
17	109.07	30	0.7	0.5	150	**0,70737 (0,09415)**	**0,68147 (0,14251)**	0,57626 (0,14942)	**0,75542 (0,01316)**	0,61138 (0,11362)	**0,97435 (0,13130)**	**0,97994 (0,08505)**
18	120.36	6	0.3	0.5	300	0,65581 (0,09544)	0,47439 (0,13851)	**0,68827 (0,14551)**	**0,75458 (0,01355)**	0,65105 (0,18962)	**0,97959 (0,08974)**	**0,97994 (0,08505)**
19	110.71	3	1.0	0.5	300	0,63900 (0,08046)	0,45898 (0,11685)	**0,63997 (0,17848)**	**0,75195 (0,02429)**	**0,76642 (0,16162)**	**0,99550 (0,02466)**	**0,98506 (0,05685)**
20	109.71	30	1.0	0.5	300	**0,71771 (0,10083)**	**0,69164 (0,16405)**	**0,66497 (0,16591)**	0,75172 (0,01636)	0,60650 (0,15930)	**0,98312 (0,07278)**	**0,93082 (0,15126)**
21	36.36	6	0.5	2.0	150	**0,73547 (0,03266)**	**0,63837 (0,06221)**	**0,74329 (0,09747)**	**0,76422 (0,00537)**	**0,75991 (0,17924)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
22	54.14	6	0.3	2.0	150	**0,74052 (0,03747)**	0,61985 (0,06108)	**0,72260 (0,11756)**	**0,76329 (0,00504)**	**0,70767 (0,14289)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
23	84.50	3	0.9	0.5	150	0,65693 (0,07012)	0,45686 (0,12575)	**0,68697 (0,14033)**	**0,75667 (0,01171)**	**0,77652 (0,14713)**	**1,00000 (0,00000)**	**0,99450 (0,03013)**
24	45.79	6	0.9	2.0	150	**0,74287 (0,04628)**	**0,63886 (0,05557)**	**0,71936 (0,09414)**	**0,76151 (0,00631)**	**0,76729 (0,13884)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
25	54.43	6	0.7	2.0	150	**0,73535 (0,04046)**	**0,65443 (0,06365)**	**0,72226 (0,10856)**	**0,76197 (0,00640)**	**0,73365 (0,14035)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
26	98.21	3	0.3	0.5	150	0,65723 (0,06760)	0,45534 (0,14744)	**0,63401 (0,15732)**	**0,75017 (0,02718)**	**0,77920 (0,16732)**	**1,00000 (0,00000)**	**0,98467 (0,05099)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
27	121.86	6	0.7	0.5	300	0,65149 (0,08695)	0,55939 (0,14285)	**0,63990 (0,16695)**	**0,74757 (0,02841)**	0,64660 (0,14679)	**0,99302 (0,02534)**	**0,97485 (0,06397)**
28	87.36	3	0.1	0.5	150	0,65886 (0,07033)	0,50350 (0,11601)	**0,67651 (0,12490)**	**0,75209 (0,02385)**	**0,76440 (0,15302)**	**1,00000 (0,00000)**	**0,99529 (0,01794)**
29	113.21	6	0.5	0.5	300	0,67274 (0,10339)	0,56166 (0,17444)	**0,66072 (0,19341)**	0,74919 (0,02411)	0,65428 (0,18053)	**0,99835 (0,00628)**	**0,97247 (0,09272)**
30	111.14	30	0.3	0.5	150	**0,70489 (0,08414)**	**0,64147 (0,14959)**	**0,67612 (0,14995)**	**0,75138 (0,03198)**	0,62580 (0,15692)	**0,98567 (0,03578)**	**0,96029 (0,09913)**
31	93.00	3	0.7	0.5	150	0,66795 (0,07830)	0,44758 (0,11219)	**0,66423 (0,14297)**	**0,75264 (0,02231)**	**0,76676 (0,16618)**	**1,00000 (0,00000)**	**0,99017 (0,04247)**
32	99.00	3	0.5	0.5	150	0,67644 (0,07487)	0,45565 (0,13562)	**0,64537 (0,14963)**	**0,75668 (0,01416)**	**0,73927 (0,17940)**	**0,99835 (0,00628)**	**0,99764 (0,01291)**
33	73.71	6	0.1	2.0	150	**0,74118 (0,04098)**	**0,62277 (0,04938)**	**0,69357 (0,12275)**	**0,76300 (0,00551)**	**0,68430 (0,15169)**	**0,99917 (0,00452)**	**0,97759 (0,06837)**
34	114.29	30	0.1	0.5	150	**0,70304 (0,07525)**	**0,63740 (0,14432)**	**0,68214 (0,15384)**	0,74686 (0,02411)	0,59674 (0,20145)	**0,99745 (0,01033)**	**0,87360 (0,23368)**
35	114.93	6	0.9	0.5	300	0,67125 (0,07528)	0,55897 (0,12650)	**0,64465 (0,16764)**	0,74903 (0,02548)	0,64262 (0,17450)	**0,98800 (0,06573)**	**0,99253 (0,04091)**
36	52.21	3	1.0	5.0	150	0,70461 (0,03026)	0,53987 (0,05174)	**0,74144 (0,10036)**	**0,76235 (0,00607)**	**0,80312 (0,13409)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
37	58.86	6	0.3	1.0	150	**0,73938 (0,04961)**	0,60630 (0,08119)	**0,71714 (0,12147)**	**0,75928 (0,01033)**	**0,78035 (0,14730)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
38	40.64	3	0.3	5.0	300	0,70847 (0,02501)	0,51793 (0,05145)	**0,75855 (0,09439)**	**0,76460 (0,00550)**	**0,80993 (0,13917)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
39	59.86	30	0.1	2.0	150	**0,75999 (0,03442)**	**0,75852 (0,07382)**	**0,71737 (0,08839)**	**0,76445 (0,00599)**	0,62883 (0,11310)	**0,99917 (0,00452)**	**0,95986 (0,14664)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
40	72.07	6	0.7	1.0	150	**0,72146 (0,04986)**	**0,62392 (0,07421)**	**0,71106 (0,12175)**	**0,75600 (0,01974)**	**0,71693 (0,18582)**	**1,00000 (0,00000)**	**0,99056 (0,05169)**
41	115.93	6	0.3	0.5	150	0,66558 (0,09624)	0,57088 (0,12105)	**0,62445 (0,14782)**	**0,74965 (0,02611)**	**0,68996 (0,16414)**	**0,99835 (0,00628)**	**0,95519 (0,11947)**
42	56.29	3	0.7	5.0	300	**0,70824 (0,03418)**	0,55387 (0,05299)	**0,75300 (0,09818)**	**0,76082 (0,00748)**	**0,79384 (0,16832)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
43	61.86	3	0.5	2.0	300	0,70470 (0,03093)	0,52762 (0,05970)	**0,73940 (0,10000)**	**0,75983 (0,00912)**	**0,82706 (0,13226)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
44	98.71	6	0.7	0.5	150	**0,69484 (0,08624)**	0,60503 (0,09803)	**0,65320 (0,17856)**	**0,75686 (0,01450)**	**0,73650 (0,16714)**	**0,99880 (0,00655)**	**0,96346 (0,12606)**
45	46.00	3	0.9	2.0	300	**0,72164 (0,03545)**	0,54719 (0,07139)	**0,74814 (0,08659)**	**0,76295 (0,00730)**	**0,82337 (0,13073)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
46	63.21	6	0.1	2.0	300	**0,73737 (0,03471)**	**0,63021 (0,07950)**	**0,70821 (0,08717)**	**0,76465 (0,00515)**	**0,74677 (0,16893)**	**0,99835 (0,00628)**	**0,97759 (0,06837)**
47	53.79	3	0.1	2.0	150	0,70523 (0,03728)	0,53369 (0,06282)	**0,74256 (0,09186)**	**0,76273 (0,00649)**	**0,76592 (0,15610)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
48	52.21	6	0.5	2.0	300	**0,74872 (0,04256)**	**0,64788 (0,08013)**	**0,72094 (0,11805)**	**0,76399 (0,00615)**	**0,69655 (0,13100)**	**1,00000 (0,00000)**	**0,98506 (0,05685)**
49	55.64	3	0.1	2.0	300	0,70613 (0,03526)	0,52127 (0,07189)	**0,74430 (0,11106)**	**0,76103 (0,00761)**	**0,79919 (0,15463)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
50	48.36	3	0.5	2.0	150	**0,71757 (0,03710)**	0,52102 (0,05243)	**0,75481 (0,08274)**	**0,76242 (0,00752)**	**0,79074 (0,14830)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
51	51.79	6	0.9	2.0	300	**0,74413 (0,03554)**	**0,63015 (0,05160)**	**0,72857 (0,08601)**	**0,76456 (0,00565)**	**0,72072 (0,15647)**	**0,99917 (0,00452)**	**0,99253 (0,04091)**
52	54.57	30	1.0	5.0	150	**0,75692 (0,03812)**	**0,72991 (0,07830)**	**0,74170 (0,11514)**	**0,76282 (0,00494)**	**0,68102 (0,13993)**	**0,99835 (0,00628)**	**0,99253 (0,04091)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
53	66.57	30	0.3	5.0	300	**0,75130 (0,03277)**	**0,76991 (0,08572)**	**0,69936 (0,13212)**	**0,76199 (0,01048)**	0,67467 (0,13358)	**0,99917 (0,00452)**	**0,96972 (0,09909)**
54	72.29	30	0.5	2.0	150	**0,74987 (0,03830)**	**0,76164 (0,07579)**	**0,71509 (0,08169)**	**0,76291 (0,01075)**	0,65763 (0,14541)	**0,99752 (0,00755)**	**0,95363 (0,10524)**
55	46.64	3	0.9	2.0	150	**0,71213 (0,03502)**	0,54556 (0,05181)	**0,73620 (0,10819)**	**0,76385 (0,00463)**	**0,77349 (0,15442)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
56	71.29	30	0.7	5.0	300	**0,76136 (0,03967)**	**0,74756 (0,06713)**	**0,71654 (0,11040)**	**0,76308 (0,00481)**	0,66942 (0,11165)	**0,99460 (0,02118)**	**0,94692 (0,11148)**
57	57.21	30	0.9	2.0	150	**0,76104 (0,02923)**	**0,75294 (0,06212)**	**0,71499 (0,08319)**	**0,76414 (0,00507)**	**0,71352 (0,15663)**	**0,99835 (0,00628)**	**0,96581 (0,11244)**
58	51.29	6	1.0	5.0	300	**0,74062 (0,02992)**	**0,63297 (0,05725)**	**0,77260 (0,07872)**	**0,76375 (0,00438)**	**0,72684 (0,15769)**	**0,99917 (0,00452)**	**0,98506 (0,05685)**
59	43.43	6	1.0	2.0	150	**0,73168 (0,03489)**	**0,65191 (0,06603)**	**0,74489 (0,08917)**	**0,76345 (0,00755)**	**0,74612 (0,15542)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
60	95.93	6	1.0	0.5	150	0,67805 (0,07028)	0,59543 (0,12978)	**0,65993 (0,17470)**	**0,75652 (0,01228)**	**0,73694 (0,16093)**	**0,99006 (0,03430)**	**0,99764 (0,01291)**
61	59.86	3	1.0	5.0	300	**0,70949 (0,04131)**	0,53957 (0,06023)	**0,73494 (0,08767)**	**0,76189 (0,00671)**	**0,78289 (0,14996)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
62	51.36	3	0.3	5.0	150	0,69900 (0,02304)	0,53575 (0,05908)	**0,73266 (0,08042)**	**0,76296 (0,00592)**	**0,81660 (0,14647)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
63	47.14	6	0.7	5.0	300	**0,74263 (0,03158)**	**0,64812 (0,05476)**	**0,74319 (0,11261)**	**0,76186 (0,00629)**	**0,75368 (0,16205)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
64	40.50	3	0.7	5.0	150	0,70694 (0,03413)	0,54249 (0,06318)	**0,76519 (0,08216)**	**0,76346 (0,00597)**	**0,81806 (0,13840)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
65	65.71	30	0.7	5.0	150	**0,75048 (0,04258)**	**0,76869 (0,06866)**	**0,72476 (0,11284)**	**0,76165 (0,00691)**	**0,72403 (0,14384)**	**0,99550 (0,02466)**	**0,96185 (0,12180)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
66	62.21	30	0.1	2.0	300	**0,74811 (0,03219)**	**0,74828 (0,07115)**	**0,72246 (0,12096)**	**0,76307 (0,00589)**	**0,72303 (0,16123)**	**0,99100 (0,03427)**	**0,98466 (0,08403)**
67	39.29	6	1.0	1.0	150	**0,74432 (0,04124)**	**0,65604 (0,06801)**	**0,73686 (0,11893)**	**0,76432 (0,00482)**	**0,76218 (0,16873)**	**1,00000 (0,00000)**	**0,99017 (0,04247)**
68	73.43	30	0.5	2.0	300	**0,75850 (0,03095)**	**0,74589 (0,06857)**	**0,68853 (0,11750)**	**0,76390 (0,00472)**	0,65764 (0,14736)	**0,99670 (0,00856)**	**0,94100 (0,14486)**
69	42.36	30	0.9	2.0	300	**0,76320 (0,03999)**	**0,74868 (0,08497)**	**0,75084 (0,08440)**	**0,76311 (0,00602)**	**0,73192 (0,14730)**	**1,00000 (0,00000)**	**0,94930 (0,10884)**
70	57.43	30	0.3	5.0	150	**0,76186 (0,03705)**	**0,76455 (0,08212)**	**0,74851 (0,10435)**	**0,76311 (0,00716)**	0,67349 (0,14503)	**0,99670 (0,00856)**	**0,96381 (0,11914)**
71	63.57	30	1.0	5.0	300	**0,75170 (0,03193)**	**0,76054 (0,06619)**	**0,71605 (0,09095)**	**0,76443 (0,00614)**	**0,67474 (0,12828)**	**0,99752 (0,00755)**	**0,95796 (0,10250)**
72	56.29	6	0.3	5.0	300	**0,72599 (0,04386)**	**0,62431 (0,05451)**	**0,73782 (0,09022)**	**0,76284 (0,00669)**	**0,76061 (0,14552)**	**1,00000 (0,00000)**	**0,97839 (0,08632)**
73	53.64	6	0.7	5.0	150	**0,73217 (0,03939)**	**0,64855 (0,05018)**	**0,72655 (0,10353)**	**0,76427 (0,00498)**	**0,70477 (0,13801)**	**0,99917 (0,00452)**	**0,99764 (0,01291)**
74	56.07	6	0.9	5.0	150	**0,73645 (0,03290)**	**0,63240 (0,05746)**	**0,74788 (0,07920)**	**0,76409 (0,00466)**	**0,70096 (0,16823)**	**0,99917 (0,00452)**	**0,99017 (0,04247)**
75	68.64	6	1.0	1.0	300	**0,73978 (0,04449)**	**0,62368 (0,08593)**	**0,68161 (0,12539)**	**0,76023 (0,00834)**	**0,72834 (0,18417)**	**1,00000 (0,00000)**	**0,98978 (0,03429)**
76	94.00	30	0.9	1.0	300	**0,72783 (0,04500)**	**0,72283 (0,07988)**	**0,68729 (0,13491)**	**0,75672 (0,01100)**	0,61977 (0,13365)	**0,99467 (0,02492)**	**0,96972 (0,09909)**
77	88.79	3	0.1	1.0	300	0,69713 (0,06120)	0,51160 (0,10351)	**0,70254 (0,12340)**	0,75362 (0,01888)	**0,74795 (0,15066)**	**0,99917 (0,00452)**	**0,99253 (0,04091)**
78	46.64	3	1.0	1.0	150	**0,70776 (0,03795)**	0,55983 (0,06099)	**0,75705 (0,09019)**	**0,76125 (0,00717)**	**0,81338 (0,13464)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
79	79.86	3	0.3	1.0	300	0,69247 (0,05981)	0,49893 (0,09314)	**0,67281 (0,13364)**	**0,76084 (0,00837)**	**0,83470 (0,15194)**	**0,99752 (0,00755)**	**1,00000 (0,00000)**
80	54.36	30	1.0	2.0	300	**0,75757 (0,04003)**	**0,76806 (0,07063)**	**0,71909 (0,10517)**	**0,76451 (0,00395)**	**0,70736 (0,16802)**	**0,99835 (0,00628)**	**0,96972 (0,09909)**
81	35.29	6	0.1	5.0	150	**0,74092 (0,03568)**	**0,65376 (0,04935)**	**0,75399 (0,10757)**	**0,76441 (0,00453)**	**0,76134 (0,15480)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
82	56.64	30	1.0	1.0	150	**0,75280 (0,03545)**	**0,75484 (0,06744)**	**0,66767 (0,13165)**	**0,76454 (0,00515)**	**0,70753 (0,16790)**	**1,00000 (0,00000)**	**0,94773 (0,12351)**
83	45.86	6	0.3	5.0	150	**0,73674 (0,02481)**	**0,66043 (0,06253)**	**0,74963 (0,10077)**	**0,76234 (0,00653)**	**0,74098 (0,16018)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
84	58.07	6	0.5	5.0	150	**0,73100 (0,03843)**	**0,64245 (0,06711)**	**0,71786 (0,11435)**	**0,76291 (0,00651)**	**0,72815 (0,14888)**	**0,99917 (0,00452)**	**0,99764 (0,01291)**
85	75.21	3	0.5	1.0	300	0,66735 (0,05526)	0,50566 (0,07832)	**0,70053 (0,11622)**	**0,75804 (0,01274)**	**0,76822 (0,16018)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
86	79.07	30	0.3	1.0	300	**0,73340 (0,05188)**	**0,70285 (0,09775)**	**0,65880 (0,11633)**	**0,75818 (0,01117)**	**0,67477 (0,12103)**	**1,00000 (0,00000)**	**0,96972 (0,09909)**
87	72.93	3	0.7	1.0	300	0,68471 (0,05267)	0,52948 (0,08601)	**0,69368 (0,12648)**	**0,76250 (0,00800)**	**0,80897 (0,15905)**	**0,99835 (0,00628)**	**1,00000 (0,00000)**
88	86.29	30	0.1	1.0	300	**0,74461 (0,05912)**	**0,68664 (0,11184)**	**0,65411 (0,14454)**	**0,75797 (0,01138)**	**0,68405 (0,16077)**	**0,99917 (0,00452)**	**0,93641 (0,19130)**
89	101.29	30	0.7	1.0	300	**0,71175 (0,08129)**	**0,71391 (0,08817)**	**0,68304 (0,12821)**	**0,75527 (0,01585)**	0,66593 (0,14633)	**0,99340 (0,02338)**	**0,93705 (0,16765)**
90	80.71	30	0.5	1.0	300	**0,75293 (0,05247)**	**0,71485 (0,07046)**	**0,70537 (0,10451)**	**0,76128 (0,00839)**	0,66602 (0,14957)	**0,99423 (0,02163)**	**0,95989 (0,10492)**
91	77.64	3	0.9	1.0	300	0,68273 (0,05009)	0,48122 (0,09526)	**0,65976 (0,12374)**	**0,76030 (0,00781)**	**0,76869 (0,15318)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
92	80.07	6	0.1	1.0	300	**0,70666 (0,06877)**	0,56683 (0,10242)	**0,68724 (0,13982)**	**0,76172 (0,01019)**	**0,69074 (0,14518)**	**0,99835 (0,00628)**	**1,00000 (0,00000)**
93	71.86	6	0.3	1.0	300	**0,70877 (0,06227)**	0,60834 (0,08689)	**0,72643 (0,10588)**	**0,76130 (0,00706)**	**0,71818 (0,15050)**	**1,00000 (0,00000)**	**0,97719 (0,09219)**
94	51.79	30	0.9	1.0	150	**0,75361 (0,05154)**	**0,73380 (0,08204)**	**0,73672 (0,11280)**	**0,76317 (0,00499)**	0,62954 (0,12609)	**1,00000 (0,00000)**	**0,98506 (0,05685)**
95	68.71	30	1.0	2.0	150	**0,75974 (0,03698)**	**0,77698 (0,07066)**	**0,70869 (0,11929)**	**0,76195 (0,01015)**	0,66249 (0,15567)	**0,99917 (0,00452)**	**0,93944 (0,13320)**
96	85.64	6	0.7	1.0	300	**0,71563 (0,05379)**	0,61008 (0,07582)	**0,65102 (0,13732)**	**0,75894 (0,01050)**	**0,71622 (0,15885)**	**0,99752 (0,00755)**	**0,99764 (0,01291)**
97	71.79	3	1.0	1.0	300	**0,70264 (0,04963)**	0,52265 (0,09240)	**0,72655 (0,12658)**	**0,76038 (0,00568)**	**0,76916 (0,14519)**	**1,00000 (0,00000)**	**0,98703 (0,04996)**
98	64.00	3	0.1	1.0	150	**0,71177 (0,05706)**	0,54126 (0,07250)	**0,72042 (0,12096)**	**0,75938 (0,00987)**	**0,79423 (0,13668)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
99	82.36	6	0.5	1.0	300	**0,70647 (0,05271)**	0,57428 (0,09107)	**0,67935 (0,10948)**	**0,75786 (0,01194)**	**0,74232 (0,15741)**	**0,99917 (0,00452)**	**0,99253 (0,04091)**
100	74.57	3	0.3	1.0	150	**0,71180 (0,04860)**	0,48818 (0,06122)	**0,67849 (0,11302)**	**0,75972 (0,00723)**	**0,76061 (0,11937)**	**1,00000 (0,00000)**	**0,99764 (0,01291)**
101	70.07	3	0.7	1.0	150	0,68706 (0,05610)	0,48398 (0,08755)	**0,68420 (0,12360)**	**0,76021 (0,00945)**	**0,82225 (0,14499)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
102	85.71	30	0.3	1.0	150	**0,73519 (0,05139)**	**0,74853 (0,06851)**	**0,69596 (0,12649)**	**0,75659 (0,01951)**	**0,71583 (0,18263)**	**0,99752 (0,00755)**	**0,89226 (0,17468)**
103	65.64	30	0.1	1.0	150	**0,74454 (0,04140)**	**0,74882 (0,08036)**	**0,69807 (0,12637)**	**0,76081 (0,00825)**	0,66960 (0,14228)	**0,99917 (0,00452)**	**0,99253 (0,04091)**
104	94.36	3	0.5	1.0	150	0,69762 (0,05075)	0,52723 (0,06857)	**0,69507 (0,11894)**	**0,75956 (0,01156)**	**0,74333 (0,14756)**	**0,99828 (0,00945)**	**0,97839 (0,08632)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
105	84.29	30	0.7	1.0	150	**0,75773 (0,04268)**	**0,73530 (0,08130)**	**0,69538 (0,12454)**	**0,76148 (0,00664)**	0,63881 (0,12105)	**0,98125 (0,10271)**	**0,95949 (0,12172)**
106	47.71	6	1.0	5.0	150	**0,74018 (0,03356)**	**0,65593 (0,05621)**	**0,74813 (0,09579)**	**0,76382 (0,00524)**	0,66348 (0,12017)	**1,00000 (0,00000)**	**0,99253 (0,04091)**
107	66.14	6	0.9	1.0	300	**0,71560 (0,06119)**	0,61125 (0,08227)	**0,73008 (0,12131)**	**0,75867 (0,00967)**	**0,72610 (0,15314)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
108	48.00	6	1.0	2.0	300	**0,74709 (0,04095)**	**0,65070 (0,05153)**	**0,74646 (0,09645)**	**0,76252 (0,00567)**	**0,70496 (0,15057)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
109	79.43	30	0.5	1.0	150	**0,74295 (0,04081)**	**0,72883 (0,08881)**	**0,68153 (0,13074)**	**0,75996 (0,01123)**	**0,70864 (0,14683)**	**0,99917 (0,00452)**	**0,93164 (0,13531)**
110	57.86	3	0.9	1.0	150	0,69698 (0,05409)	0,50652 (0,07640)	**0,74174 (0,10920)**	**0,76338 (0,00645)**	**0,81004 (0,13806)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**
111	48.21	3	1.0	2.0	150	0,70428 (0,03532)	0,49750 (0,04540)	**0,77268 (0,09927)**	**0,76291 (0,00660)**	**0,81348 (0,16706)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
112	76.14	30	1.0	1.0	300	**0,74838 (0,05685)**	**0,70149 (0,08287)**	**0,67087 (0,13389)**	**0,76411 (0,00517)**	0,67173 (0,14273)	**0,99467 (0,02492)**	**0,96932 (0,11677)**
113	57.00	6	0.5	1.0	150	**0,72053 (0,05057)**	0,61694 (0,07557)	**0,74474 (0,09468)**	**0,75852 (0,01256)**	**0,76364 (0,15940)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
114	113.43	6	0.1	0.5	150	0,67655 (0,08376)	0,55563 (0,11980)	**0,63642 (0,16332)**	**0,75157 (0,02121)**	**0,67775 (0,17022)**	**0,99175 (0,03155)**	**0,98506 (0,05685)**
115	54.79	6	0.9	1.0	150	**0,71557 (0,04760)**	**0,64123 (0,06367)**	**0,74895 (0,09235)**	**0,76056 (0,00806)**	**0,77918 (0,14012)**	**1,00000 (0,00000)**	**0,98351 (0,07806)**
116	116.36	6	0.5	0.5	150	0,67995 (0,08846)	0,56485 (0,12971)	**0,64147 (0,15964)**	0,74427 (0,04097)	0,65514 (0,12852)	**0,97991 (0,07028)**	**0,99017 (0,04247)**
117	56.29	3	0.5	5.0	300	**0,70977 (0,03096)**	0,55129 (0,05629)	**0,73345 (0,09633)**	**0,76299 (0,00534)**	**0,79327 (0,12203)**	**1,00000 (0,00000)**	**0,99017 (0,04247)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
118	90.71	6	0.9	0.5	150	0,68239 (0,07392)	0,56998 (0,10397)	**0,69983 (0,14839)**	0,74993 (0,02207)	**0,72275 (0,14121)**	**0,99550 (0,02466)**	**1,00000 (0,00000)**
119	57.57	3	0.7	2.0	300	0,69376 (0,03824)	0,52487 (0,06401)	**0,76802 (0,10510)**	**0,76227 (0,00694)**	**0,81122 (0,15580)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
120	70.00	6	0.1	1.0	150	**0,72289 (0,05270)**	0,61825 (0,07580)	**0,74465 (0,11350)**	**0,75906 (0,01126)**	**0,72238 (0,15028)**	**0,99917 (0,00452)**	**0,98506 (0,05685)**
121	64.36	3	0.1	5.0	300	0,70243 (0,02695)	0,52551 (0,05220)	**0,73713 (0,08340)**	**0,76257 (0,00631)**	**0,79821 (0,14743)**	**0,99835 (0,00628)**	**1,00000 (0,00000)**
122	53.43	6	0.3	2.0	300	**0,74099 (0,04031)**	**0,62433 (0,05404)**	**0,73115 (0,10198)**	**0,76264 (0,00852)**	**0,72123 (0,13861)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
123	50.00	3	0.3	2.0	150	0,70371 (0,03352)	0,53445 (0,06584)	**0,75572 (0,07826)**	**0,76296 (0,00689)**	**0,85117 (0,11176)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
124	72.29	30	0.1	5.0	300	**0,76783 (0,03584)**	**0,73519 (0,06689)**	**0,68710 (0,10871)**	**0,76319 (0,00546)**	**0,69803 (0,16769)**	**0,99467 (0,02492)**	**0,94573 (0,16225)**
125	45.50	3	0.3	2.0	300	**0,70849 (0,03280)**	0,52773 (0,06634)	**0,74721 (0,09142)**	**0,76347 (0,00519)**	**0,79889 (0,14792)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
126	59.00	6	0.7	2.0	300	**0,74378 (0,04723)**	**0,64023 (0,05719)**	**0,71981 (0,10620)**	**0,76239 (0,00638)**	0,67574 (0,13466)	**0,99917 (0,00452)**	**1,00000 (0,00000)**
127	54.14	30	0.5	5.0	300	**0,74863 (0,04404)**	**0,74420 (0,08271)**	**0,74004 (0,08101)**	**0,76393 (0,00566)**	**0,70754 (0,18282)**	**0,99917 (0,00452)**	**0,96736 (0,09919)**
128	52.93	3	0.9	5.0	300	0,70119 (0,03855)	0,54233 (0,04554)	**0,71392 (0,10687)**	**0,76313 (0,00708)**	**0,82630 (0,14403)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
129	70.29	30	0.3	2.0	150	**0,75197 (0,03657)**	**0,75256 (0,06776)**	**0,70829 (0,12989)**	**0,75956 (0,00830)**	**0,69212 (0,14602)**	**0,99670 (0,01075)**	**0,98586 (0,07743)**
130	46.14	3	0.7	2.0	150	**0,71712 (0,03043)**	0,53088 (0,06479)	**0,74431 (0,09238)**	**0,76524 (0,00368)**	**0,80682 (0,15845)**	**0,99917 (0,00452)**	**1,00000 (0,00000)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	Nr	D	C	W	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
131	72.93	30	0.7	2.0	150	**0,75799 (0,03233)**	**0,73521 (0,07503)**	**0,68539 (0,13254)**	**0,76317 (0,00677)**	0,61733 (0,10577)	**0,99835 (0,00628)**	**0,96972 (0,09909)**
132	71.57	30	0.9	5.0	300	**0,73702 (0,03398)**	**0,76605 (0,08224)**	**0,71744 (0,10232)**	**0,76411 (0,00493)**	0,64486 (0,12937)	**0,99430 (0,02529)**	**0,96541 (0,07767)**
133	47.43	3	0.1	5.0	150	**0,71905 (0,03423)**	0,53145 (0,05533)	**0,73425 (0,08600)**	**0,76449 (0,00530)**	**0,81262 (0,13198)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
134	61.07	6	0.5	5.0	300	**0,73420 (0,02740)**	**0,63467 (0,05417)**	**0,72127 (0,12466)**	**0,76244 (0,00764)**	**0,74556 (0,17477)**	**1,00000 (0,00000)**	**0,97093 (0,09375)**
135	53.07	6	0.1	5.0	300	**0,73850 (0,04097)**	**0,65230 (0,05700)**	**0,73060 (0,10595)**	**0,76285 (0,00634)**	**0,75051 (0,15240)**	**1,00000 (0,00000)**	**0,97759 (0,06837)**
136	42.50	3	0.5	5.0	150	**0,71793 (0,03952)**	0,53203 (0,05825)	**0,73535 (0,08872)**	**0,76360 (0,00644)**	**0,85604 (0,12172)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
137	59.64	6	0.9	5.0	300	**0,72696 (0,03294)**	**0,62802 (0,04671)**	**0,74530 (0,09851)**	**0,76232 (0,00583)**	**0,72024 (0,15670)**	**1,00000 (0,00000)**	**0,97839 (0,08632)**
138	69.71	30	0.3	2.0	300	**0,75559 (0,03786)**	**0,74385 (0,06750)**	**0,70409 (0,08983)**	**0,76241 (0,01047)**	0,66084 (0,13741)	**0,99752 (0,00755)**	**0,98506 (0,05685)**
139	51.14	30	0.5	5.0	150	**0,76135 (0,03508)**	**0,74428 (0,07323)**	**0,69253 (0,10094)**	**0,76392 (0,00507)**	**0,70760 (0,16485)**	**1,00000 (0,00000)**	**0,97719 (0,09219)**
140	43.71	3	0.9	5.0	150	**0,71686 (0,04090)**	0,53138 (0,05770)	**0,74752 (0,09547)**	**0,76432 (0,00494)**	**0,86990 (0,13279)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**
141	67.57	30	0.7	2.0	300	**0,75024 (0,04085)**	**0,74722 (0,07852)**	**0,70832 (0,11548)**	**0,76262 (0,00698)**	**0,70800 (0,14038)**	**0,99917 (0,00452)**	**0,93357 (0,15550)**
142	60.93	30	0.9	5.0	150	**0,75411 (0,03023)**	**0,75336 (0,06607)**	**0,71542 (0,08883)**	**0,76550 (0,00444)**	0,67471 (0,13189)	**0,99835 (0,00628)**	**0,95635 (0,12373)**
143	55.36	30	0.1	5.0	150	**0,74720 (0,03435)**	**0,75725 (0,08328)**	**0,72619 (0,11172)**	**0,76467 (0,00510)**	**0,69140 (0,18028)**	**0,99752 (0,00996)**	**0,98466 (0,08403)**

Continua na próxima página

Tabela A.1 – continuação dos resultados de *tuning* para configurações do FRRMAB

Id	Rank	**Nr**	**D**	**C**	**W**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
144	46.86	3	1.0	2.0	300	**0,72054 (0,02259)**	0,53247 (0,04597)	**0,76772 (0,10321)**	**0,76271 (0,00636)**	**0,81891 (0,15009)**	**1,00000 (0,00000)**	**0,99253 (0,04091)**

Tabela A.2: Resultados de *tuning* para configurações do FRRCF

Id	Rank	β	**Nr**	**D**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
1	28.43	0.10	3	0.3	**0,78900 (0,02848)**	0,82166 (0,01999)	**0,90118 (0,03731)**	**0,82255 (0,00696)**	**0,95094 (0,03640)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
2	76.21	0.00005	6	0.3	0,74237 (0,04488)	0,84445 (0,03165)	**0,87370 (0,04073)**	0,81677 (0,00793)	**0,93715 (0,03389)**	**1,00000 (0,00000)**	**0,98798 (0,04696)**
3	80.71	0.001	30	0.1	**0,71255 (0,17352)**	0,71960 (0,26370)	0,67162 (0,25886)	0,76526 (0,09340)	**0,87538 (0,11714)**	**0,92270 (0,12815)**	**0,94170 (0,12686)**
4	47.50	0.01	6	0.5	0,75006 (0,03616)	0,83897 (0,03680)	**0,87339 (0,04039)**	**0,81936 (0,00627)**	**0,94813 (0,03384)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
5	75.71	0.00005	3	0.5	0,72224 (0,04456)	0,77872 (0,04058)	0,79093 (0,04963)	**0,81657 (0,00811)**	**0,93743 (0,02923)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
6	22.50	0.05	30	1.0	**0,82674 (0,02225)**	**0,91430 (0,03117)**	**0,89242 (0,03886)**	**0,82320 (0,00617)**	**0,92014 (0,03782)**	**0,99970 (0,00164)**	**0,99194 (0,01980)**
7	70.86	0.005	3	0.1	0,72467 (0,03999)	0,77433 (0,02845)	0,79596 (0,04579)	0,81630 (0,00818)	**0,92833 (0,03599)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
8	64.71	0.005	3	0.7	0,70656 (0,02186)	0,76403 (0,02600)	0,81040 (0,04769)	**0,81828 (0,00733)**	**0,95021 (0,02984)**	**1,00000 (0,00000)**	**0,99998 (0,00010)**
9	68.79	0.00005	6	0.9	0,76621 (0,04360)	0,84333 (0,03370)	**0,87164 (0,05094)**	0,81566 (0,00897)	**0,92886 (0,07533)**	**0,99972 (0,00156)**	**1,00000 (0,00000)**
10	79.71	0.001	3	1.0	0,72361 (0,06848)	0,77278 (0,07520)	0,77200 (0,12979)	**0,80376 (0,04038)**	**0,91355 (0,07180)**	**0,99192 (0,02061)**	**0,99353 (0,01883)**
11	27.00	0.10	6	0.1	**0,80108 (0,02788)**	**0,86591 (0,02402)**	**0,89764 (0,04588)**	**0,82229 (0,00593)**	**0,93252 (0,03173)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
12	58.64	0.001	6	0.7	0,74515 (0,05690)	0,83606 (0,06639)	**0,87365 (0,03505)**	**0,81803 (0,00664)**	**0,93211 (0,03115)**	**0,99972 (0,00156)**	**0,99844 (0,00854)**
13	37.71	0.05	3	0.5	0,75827 (0,02214)	0,79885 (0,03070)	**0,88387 (0,03949)**	**0,82239 (0,00521)**	**0,94703 (0,02972)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	Nr	D	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
14	28.57	0.10	3	1.0	**0,78319 (0,02861)**	0,81549 (0,02322)	**0,90571 (0,03230)**	**0,82408 (0,00502)**	**0,95651 (0,02632)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
15	92.57	0.00005	30	0.1	0,59846 (0,28733)	**0,69039 (0,37996)**	**0,67203 (0,37008)**	0,70625 (0,17433)	0,67640 (0,34757)	**0,75322 (0,34703)**	**0,73214 (0,36835)**
16	97.71	0.00005	30	0.7	**0,65237 (0,27489)**	**0,72788 (0,36201)**	**0,59576 (0,41809)**	0,73650 (0,14783)	0,62646 (0,32535)	**0,81146 (0,30743)**	**0,75692 (0,36503)**
17	74.57	0.001	3	0.3	0,70229 (0,02634)	0,77668 (0,03299)	0,78906 (0,05841)	0,81373 (0,00838)	**0,93161 (0,02748)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
18	64.36	0.005	30	1.0	0,74043 (0,08404)	**0,89277 (0,06970)**	**0,82449 (0,10636)**	0,78998 (0,05020)	0,89997 (0,03927)	**0,98433 (0,05220)**	**0,99192 (0,01497)**
19	31.29	0.10	30	0.3	**0,82463 (0,03061)**	**0,90540 (0,02028)**	**0,88270 (0,03647)**	**0,82489 (0,00541)**	0,91615 (0,03080)	**0,99972 (0,00156)**	**0,99731 (0,00873)**
20	30.21	0.05	6	0.9	**0,80619 (0,02116)**	**0,87499 (0,02563)**	**0,87585 (0,04138)**	**0,82289 (0,00571)**	**0,94961 (0,03348)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
21	52.50	0.001	6	0.1	0,75622 (0,04143)	0,85125 (0,03925)	**0,84693 (0,11223)**	0,81320 (0,01728)	**0,92837 (0,03516)**	**1,00000 (0,00000)**	**0,99735 (0,01451)**
22	29.71	0.10	6	0.7	**0,80860 (0,02715)**	**0,87354 (0,02167)**	**0,88182 (0,04412)**	**0,82290 (0,00615)**	**0,93624 (0,03700)**	**1,00000 (0,00000)**	**0,99553 (0,01158)**
23	74.43	0.001	3	0.9	0,71324 (0,04083)	0,76906 (0,03484)	0,78923 (0,05029)	0,81611 (0,00763)	**0,93443 (0,03170)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
24	29.00	0.05	6	0.3	**0,79173 (0,02301)**	**0,86150 (0,02871)**	**0,90090 (0,03576)**	**0,82381 (0,00489)**	**0,94055 (0,03600)**	**1,00000 (0,00000)**	**0,99777 (0,00850)**
25	33.36	0.10	30	1.0	**0,82240 (0,02617)**	**0,91672 (0,02588)**	**0,89685 (0,04495)**	**0,82398 (0,00540)**	**0,92517 (0,03577)**	**1,00000 (0,00000)**	**0,99418 (0,01911)**
26	35.07	0.10	3	0.9	**0,78824 (0,01809)**	0,81818 (0,02692)	**0,91314 (0,04021)**	**0,82401 (0,00516)**	**0,96147 (0,03044)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	Nr	D	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
27	31.50	0.05	6	1.0	**0,81789 (0,02630)**	**0,86578 (0,02179)**	**0,87746 (0,05387)**	**0,82288 (0,00652)**	**0,94257 (0,03841)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
28	53.64	0.01	30	0.5	**0,77201 (0,06794)**	**0,87694 (0,07073)**	**0,86526 (0,08741)**	**0,80795 (0,03488)**	0,91066 (0,03710)	**0,99824 (0,00545)**	**0,99666 (0,01593)**
29	77.14	0.001	30	0.7	**0,74663 (0,13840)**	**0,78562 (0,24595)**	**0,70223 (0,25619)**	**0,77671 (0,10396)**	0,84948 (0,12529)	**0,93254 (0,12437)**	**0,94623 (0,11808)**
30	36.50	0.10	30	0.9	**0,80422 (0,02589)**	**0,91063 (0,02180)**	**0,88676 (0,05260)**	**0,82295 (0,00691)**	**0,92656 (0,03523)**	**0,99972 (0,00156)**	**0,99777 (0,00850)**
31	58.86	0.005	30	0.9	**0,76294 (0,07955)**	**0,88332 (0,08431)**	**0,84856 (0,10428)**	**0,80733 (0,05263)**	**0,91691 (0,04367)**	**0,99190 (0,01623)**	**0,99350 (0,01744)**
32	60.43	0.005	30	0.3	**0,76184 (0,08040)**	**0,84850 (0,10651)**	**0,84441 (0,10468)**	**0,80656 (0,03467)**	**0,90677 (0,06032)**	**0,99481 (0,01261)**	**0,97948 (0,04836)**
33	56.36	0.005	6	0.7	0,76504 (0,04211)	0,84551 (0,03056)	**0,87229 (0,04373)**	**0,81729 (0,00840)**	**0,94121 (0,03188)**	**0,99972 (0,00156)**	**1,00000 (0,00000)**
34	50.50	0.005	30	0.5	**0,75520 (0,08311)**	**0,86827 (0,08037)**	0,78666 (0,11748)	**0,79310 (0,06919)**	0,89639 (0,05794)	**0,99325 (0,01617)**	**0,98839 (0,02634)**
35	65.43	0.01	3	0.5	0,71750 (0,02621)	0,77141 (0,02661)	0,80681 (0,04261)	**0,81889 (0,00584)**	**0,94056 (0,03348)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
36	41.93	0.10	30	0.7	**0,82680 (0,02726)**	**0,91910 (0,02711)**	**0,88674 (0,05471)**	**0,82261 (0,00596)**	**0,92210 (0,03877)**	**0,99943 (0,00217)**	**0,99487 (0,01705)**
37	61.64	0.00005	6	0.5	0,76294 (0,05117)	0,82022 (0,10424)	**0,86098 (0,11335)**	0,81283 (0,02042)	**0,93477 (0,03892)**	**0,99311 (0,02211)**	**1,00000 (0,00000)**
38	81.00	0.001	30	0.3	**0,73504 (0,16261)**	**0,78922 (0,21946)**	**0,73563 (0,25292)**	**0,78315 (0,07207)**	0,82058 (0,13743)	**0,91595 (0,13725)**	**0,99194 (0,02135)**
39	51.79	0.005	6	0.1	0,75542 (0,02860)	0,84825 (0,04190)	**0,88417 (0,03911)**	**0,81648 (0,00988)**	**0,94252 (0,03326)**	**0,99762 (0,00741)**	**0,99888 (0,00612)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	**Nr**	**D**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
40	67.57	0.005	3	0.3	0,71649 (0,03125)	0,78145 (0,03152)	0,79562 (0,04896)	0,81535 (0,00799)	**0,94112 (0,03162)**	**1,00000 (0,00000)**	**0,99955 (0,00248)**
41	74.86	0.001	30	0.9	**0,75469 (0,13922)**	**0,77423 (0,23678)**	0,68096 (0,25337)	**0,79773 (0,06507)**	0,87255 (0,10494)	**0,94963 (0,10081)**	**0,96035 (0,07778)**
42	47.14	0.01	6	0.3	**0,76907 (0,04269)**	0,85077 (0,03540)	**0,88429 (0,03021)**	**0,81826 (0,00744)**	**0,94395 (0,03430)**	**1,00000 (0,00000)**	**0,99955 (0,00248)**
43	69.14	0.005	3	0.9	0,72177 (0,03228)	0,77118 (0,02225)	0,80595 (0,05485)	**0,81671 (0,00753)**	**0,94512 (0,03037)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
44	51.07	0.01	30	0.7	**0,79103 (0,05349)**	**0,88543 (0,05904)**	**0,86728 (0,05283)**	**0,80932 (0,02549)**	0,91169 (0,03493)	**0,99791 (0,00615)**	**0,99620 (0,01037)**
45	77.43	0.00005	3	0.1	0,72076 (0,03255)	0,77101 (0,04280)	0,80116 (0,06021)	0,81106 (0,01068)	**0,93406 (0,02436)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
46	56.93	0.01	30	0.1	**0,77340 (0,06396)**	**0,88303 (0,05683)**	**0,84677 (0,07393)**	**0,81289 (0,02178)**	**0,92525 (0,04282)**	**0,99724 (0,01061)**	**0,98690 (0,05183)**
47	71.00	0.00005	3	0.7	0,70766 (0,02610)	0,77061 (0,03327)	0,77833 (0,06223)	0,81487 (0,00716)	**0,93014 (0,02632)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
48	31.93	0.10	30	0.1	**0,82366 (0,03654)**	**0,92447 (0,02743)**	**0,87669 (0,04418)**	**0,82506 (0,00421)**	**0,92461 (0,03611)**	**0,99943 (0,00217)**	**0,99306 (0,01972)**
49	21.43	0.10	3	0.5	**0,79532 (0,02945)**	0,82925 (0,02386)	**0,89378 (0,03985)**	**0,82243 (0,00725)**	**0,94880 (0,03541)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
50	22.71	0.05	3	0.9	**0,78826 (0,02111)**	0,82309 (0,02760)	**0,90208 (0,04039)**	**0,82446 (0,00483)**	**0,95723 (0,03342)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
51	32.57	0.05	6	0.1	**0,78931 (0,03522)**	**0,85621 (0,03816)**	**0,89775 (0,04003)**	**0,82388 (0,00410)**	**0,94264 (0,03671)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
52	74.57	0.001	3	0.5	0,71588 (0,02435)	0,77943 (0,03621)	0,78749 (0,05363)	0,81488 (0,01069)	**0,93470 (0,02761)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	Nr	D	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
53	25.29	0.05	30	0.9	**0,82874 (0,03460)**	**0,91895 (0,02411)**	**0,87527 (0,05767)**	**0,82311 (0,00484)**	**0,92174 (0,03829)**	**0,99972 (0,00156)**	**0,99575 (0,01049)**
54	86.86	0.00005	6	1.0	0,72148 (0,13523)	0,64352 (0,24507)	0,70619 (0,27178)	**0,79838 (0,04649)**	**0,87288 (0,13319)**	**0,91990 (0,11384)**	**0,94379 (0,09109)**
55	29.86	0.05	30	0.3	**0,81089 (0,02776)**	**0,91238 (0,03470)**	**0,87835 (0,05794)**	**0,82162 (0,00752)**	**0,92964 (0,03840)**	**0,99972 (0,00156)**	**0,99777 (0,00850)**
56	25.29	0.05	6	0.7	**0,79969 (0,02745)**	**0,86631 (0,02147)**	**0,88476 (0,04660)**	**0,82448 (0,00485)**	**0,94485 (0,03851)**	**0,99972 (0,00156)**	**1,00000 (0,00000)**
57	44.14	0.05	3	0.3	0,76041 (0,02891)	0,78933 (0,03794)	**0,87362 (0,04443)**	**0,82431 (0,00448)**	**0,95146 (0,03531)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
58	26.07	0.10	6	0.9	**0,80949 (0,02589)**	**0,86578 (0,02712)**	**0,89071 (0,04732)**	**0,82408 (0,00377)**	**0,93410 (0,03820)**	**0,99972 (0,00156)**	**1,00000 (0,00000)**
59	20.29	0.10	6	0.3	**0,80892 (0,02582)**	**0,86937 (0,02646)**	**0,88986 (0,04399)**	**0,82544 (0,00319)**	**0,93758 (0,03734)**	**1,00000 (0,00000)**	**0,99777 (0,00850)**
60	62.64	0.001	6	0.9	0,76220 (0,03746)	0,84269 (0,02457)	**0,87015 (0,04039)**	0,81571 (0,00741)	**0,92472 (0,05827)**	**0,99915 (0,00260)**	**0,99138 (0,03751)**
61	59.07	0.001	6	0.3	0,75900 (0,04144)	0,84594 (0,05492)	**0,85580 (0,04906)**	0,81283 (0,02250)	**0,92873 (0,04062)**	**0,99886 (0,00488)**	**0,99843 (0,00652)**
62	26.86	0.05	3	1.0	**0,78421 (0,02523)**	0,81277 (0,02046)	**0,89989 (0,03866)**	**0,82528 (0,00425)**	**0,96056 (0,03034)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
63	25.64	0.10	30	0.5	**0,82784 (0,02821)**	**0,92174 (0,02476)**	**0,87786 (0,03497)**	**0,82486 (0,00440)**	**0,92092 (0,03338)**	**0,99972 (0,00156)**	**0,99443 (0,01762)**
64	51.21	0.005	6	1.0	**0,76126 (0,06899)**	0,83026 (0,06062)	**0,82871 (0,09288)**	**0,81031 (0,02678)**	**0,91993 (0,04234)**	**0,99715 (0,00934)**	**0,99955 (0,00248)**
65	33.64	0.05	30	0.5	**0,80609 (0,02883)**	**0,90706 (0,03435)**	**0,87854 (0,04658)**	**0,82350 (0,00947)**	**0,93402 (0,03925)**	**0,99972 (0,00156)**	**0,99910 (0,00344)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	Nr	D	MyBatis	AJHsqldb	AJHotDraw	BCEL	JHotDraw	Health-Watcher	JBoss
66	59.43	0.01	30	0.3	**0,77527 (0,06049)**	**0,88482 (0,05315)**	**0,83602 (0,08097)**	**0,81510 (0,02643)**	**0,91484 (0,05370)**	**0,99512 (0,01295)**	**0,99128 (0,02226)**
67	55.43	0.01	30	0.9	**0,79968 (0,05488)**	**0,89164 (0,04536)**	**0,84371 (0,08440)**	**0,81006 (0,02831)**	0,91589 (0,03352)	**0,99660 (0,00927)**	**0,99508 (0,01166)**
68	47.00	0.005	6	0.3	0,76649 (0,04928)	0,83615 (0,02818)	**0,87291 (0,05431)**	**0,81914 (0,00814)**	**0,94652 (0,03655)**	**0,99824 (0,00711)**	**1,00000 (0,00000)**
69	55.64	0.01	6	0.9	**0,77348 (0,04391)**	0,84232 (0,03353)	**0,87991 (0,04178)**	**0,82085 (0,00803)**	**0,93971 (0,03577)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
70	64.29	0.01	3	0.3	0,72896 (0,04307)	0,76900 (0,02993)	0,79983 (0,05229)	**0,81955 (0,00733)**	**0,94653 (0,03433)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
71	97.71	0.00005	30	0.9	0,61489 (0,28086)	0,51984 (0,42887)	**0,66950 (0,36996)**	0,72842 (0,15141)	**0,84802 (0,22182)**	**0,91179 (0,20880)**	**0,75206 (0,35249)**
72	78.14	0.00005	3	1.0	0,71205 (0,08024)	0,73062 (0,12545)	0,75776 (0,18130)	**0,81084 (0,02922)**	**0,90757 (0,08229)**	**0,98960 (0,02556)**	**0,96386 (0,08213)**
73	75.50	0.00005	6	0.7	**0,77446 (0,04872)**	0,83304 (0,06724)	**0,85051 (0,10222)**	**0,81687 (0,00794)**	**0,94320 (0,03649)**	**0,99972 (0,00156)**	**0,98221 (0,05579)**
74	100.71	0.00005	30	0.3	0,56458 (0,28437)	**0,73036 (0,34663)**	**0,69952 (0,36778)**	0,65187 (0,18997)	0,73980 (0,31641)	**0,71243 (0,36470)**	**0,82342 (0,33310)**
75	29.71	0.10	6	0.5	**0,82098 (0,02419)**	**0,87622 (0,02756)**	**0,89188 (0,04049)**	**0,82272 (0,00544)**	**0,93912 (0,03906)**	**0,99910 (0,00495)**	**0,99205 (0,02662)**
76	71.57	0.005	3	0.5	0,72279 (0,03515)	0,77721 (0,04188)	0,80428 (0,06443)	0,81478 (0,00920)	**0,93763 (0,03086)**	**1,00000 (0,00000)**	**0,99777 (0,00850)**
77	67.64	0.01	3	0.9	0,72541 (0,03578)	0,78623 (0,04647)	0,80855 (0,05502)	0,81662 (0,00742)	**0,94049 (0,02972)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
78	41.79	0.01	30	1.0	**0,77587 (0,07232)**	**0,89463 (0,05507)**	**0,86446 (0,08778)**	**0,81918 (0,01086)**	0,89821 (0,03679)	**1,00000 (0,00000)**	**0,99776 (0,00734)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	**Nr**	**D**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
79	45.57	0.01	6	0.1	0,75761 (0,03132)	**0,85529 (0,03089)**	**0,89062 (0,03358)**	**0,81926 (0,00824)**	**0,92846 (0,03721)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
80	52.57	0.005	6	0.9	0,76092 (0,03557)	0,84914 (0,03649)	**0,87191 (0,04948)**	**0,81790 (0,00647)**	**0,94436 (0,02899)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
81	47.29	0.01	6	0.7	0,76369 (0,03778)	**0,85321 (0,03690)**	**0,88717 (0,03657)**	**0,81869 (0,00797)**	**0,93170 (0,03576)**	**0,99972 (0,00156)**	**1,00000 (0,00000)**
82	61.14	0.00005	6	0.1	0,75397 (0,07178)	0,84729 (0,05044)	**0,84216 (0,11470)**	0,81679 (0,00657)	**0,93806 (0,03126)**	**0,99709 (0,01594)**	**0,99843 (0,00652)**
83	76.00	0.00005	3	0.3	0,71204 (0,03531)	0,78389 (0,04622)	0,77146 (0,11763)	0,81483 (0,00913)	**0,93595 (0,02815)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
84	77.00	0.00005	3	0.9	0,71396 (0,03885)	0,78108 (0,05599)	0,77264 (0,04348)	0,81305 (0,00801)	**0,93565 (0,02831)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
85	48.93	0.01	3	1.0	0,75560 (0,04381)	0,81467 (0,04795)	**0,84945 (0,06589)**	**0,81459 (0,01622)**	**0,94531 (0,03780)**	**0,99972 (0,00156)**	**0,99843 (0,00652)**
86	79.29	0.001	30	1.0	**0,72143 (0,17228)**	**0,73984 (0,25069)**	0,59751 (0,25967)	**0,78129 (0,09569)**	0,82020 (0,17142)	**0,97756 (0,05066)**	**0,94223 (0,10051)**
87	30.79	0.05	30	0.1	**0,80932 (0,03464)**	**0,92142 (0,03120)**	**0,87960 (0,04799)**	**0,82356 (0,00563)**	**0,91977 (0,04004)**	**0,99972 (0,00156)**	**0,99665 (0,01022)**
88	43.29	0.05	3	0.1	0,76400 (0,02967)	0,79588 (0,03362)	**0,88729 (0,03748)**	**0,82167 (0,00570)**	**0,95805 (0,03051)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
89	97.71	0.00005	30	0.5	**0,71159 (0,21610)**	**0,71030 (0,35683)**	**0,67488 (0,37895)**	0,74240 (0,14474)	0,67174 (0,34339)	**0,74062 (0,35823)**	**0,79725 (0,32992)**
90	56.64	0.001	6	0.5	0,76413 (0,04161)	0,82437 (0,04318)	**0,86875 (0,04757)**	0,81388 (0,01819)	**0,93822 (0,03416)**	**0,99972 (0,00156)**	**0,99508 (0,01166)**
91	52.29	0.005	3	1.0	0,73614 (0,05015)	0,79121 (0,07106)	0,82066 (0,08177)	**0,81858 (0,00956)**	**0,93977 (0,04714)**	**0,99843 (0,00861)**	**1,00000 (0,00000)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	**Nr**	**D**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
92	25.43	0.10	6	1.0	**0,80789 (0,02596)**	**0,86562 (0,02278)**	**0,90034 (0,03144)**	**0,82340 (0,00621)**	**0,94068 (0,03343)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
93	38.07	0.05	30	0.7	**0,81365 (0,02828)**	**0,91650 (0,02454)**	**0,88346 (0,04552)**	**0,82389 (0,00429)**	**0,92056 (0,03829)**	**1,00000 (0,00000)**	**0,99553 (0,01158)**
94	33.00	0.05	3	0.7	**0,76949 (0,02408)**	0,81080 (0,02795)	**0,89063 (0,04787)**	**0,82507 (0,00318)**	**0,94196 (0,03240)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
95	71.57	0.001	3	0.1	0,70888 (0,02559)	0,77647 (0,03978)	0,77429 (0,05975)	0,81364 (0,00812)	**0,93936 (0,02659)**	**1,00000 (0,00000)**	**0,99799 (0,00881)**
96	22.86	0.05	6	0.5	**0,79551 (0,02327)**	**0,86117 (0,02401)**	**0,91170 (0,03710)**	**0,82324 (0,00456)**	**0,92817 (0,03885)**	**1,00000 (0,00000)**	**0,99818 (0,00995)**
97	75.14	0.001	3	0.7	0,71065 (0,02609)	0,77521 (0,02721)	0,80112 (0,05848)	0,80993 (0,00829)	**0,92934 (0,02624)**	**1,00000 (0,00000)**	**0,98821 (0,06455)**
98	28.43	0.10	3	0.1	**0,76793 (0,02472)**	0,81772 (0,02682)	**0,88650 (0,04864)**	**0,82202 (0,00680)**	**0,95634 (0,03289)**	**1,00000 (0,00000)**	**0,99955 (0,00248)**
99	23.14	0.10	3	0.7	**0,79302 (0,02771)**	0,83444 (0,02971)	**0,89247 (0,03976)**	**0,82443 (0,00484)**	**0,95573 (0,03471)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
100	98.29	0.00005	30	1.0	0,62551 (0,27106)	**0,67566 (0,37658)**	**0,70931 (0,35082)**	**0,76440 (0,12274)**	0,70508 (0,35174)	**0,73183 (0,37664)**	**0,68356 (0,37266)**
101	63.21	0.005	30	0.1	**0,75215 (0,09073)**	**0,85022 (0,09538)**	**0,82819 (0,09566)**	**0,81110 (0,02032)**	0,90660 (0,04897)	**0,99568 (0,01618)**	**0,98948 (0,02769)**
102	62.86	0.005	30	0.7	**0,76728 (0,07716)**	**0,86151 (0,07731)**	**0,83683 (0,10883)**	**0,80248 (0,04796)**	**0,91542 (0,05954)**	**0,99253 (0,01740)**	**0,99524 (0,01421)**
103	80.14	0.001	6	1.0	0,71621 (0,10137)	0,79090 (0,13317)	**0,78617 (0,17716)**	0,78840 (0,06098)	**0,90356 (0,06498)**	**0,97454 (0,03800)**	**0,98834 (0,03374)**
104	35.00	0.01	6	1.0	**0,76463 (0,05006)**	**0,85418 (0,04337)**	**0,87656 (0,06413)**	**0,81683 (0,01141)**	**0,92104 (0,03783)**	**0,99828 (0,00570)**	**0,99843 (0,00652)**

Continua na próxima página

Tabela A.2 – continuação dos resultados de *tuning* para configurações do FRRCF

Id	Ranking	β	**Nr**	**D**	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health-Watcher**	**JBoss**
105	77.43	0.001	30	0.5	0,63400 (0,18846)	**0,72464 (0,26052)**	**0,79819 (0,20585)**	**0,77766 (0,07593)**	0,82427 (0,14504)	**0,94087 (0,09453)**	**0,94168 (0,10855)**
106	58.14	0.005	6	0.5	0,75257 (0,04057)	0,84957 (0,03260)	**0,88548 (0,03979)**	0,81528 (0,00948)	**0,95512 (0,03171)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
107	60.71	0.01	3	0.7	0,71752 (0,02422)	0,78232 (0,03454)	0,81147 (0,04568)	**0,81670 (0,00852)**	**0,94299 (0,02587)**	**1,00000 (0,00000)**	**0,99888 (0,00612)**
108	65.71	0.01	3	0.1	0,73094 (0,03578)	0,78836 (0,04231)	0,81051 (0,05665)	**0,81807 (0,00718)**	**0,94253 (0,03505)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**

Tabela A.3: Teste de parâmetros para a HITO-NSGA-CF

Alg	α	β	Rank	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health Watcher**	**JBoss**
NSGAII_0.10	1	0.10	5.2857	0,66699 (0,02135)	0,48539 (0,07183)	0,60021 (0,11893)	0,61690 (0,00235)	**0,56916 (0,32273)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
NSGAII_0.05	1	0.05	4.1428	0,68429 (0,02332)	0,51761 (0,06011)	**0,66640 (0,14520)**	0,61691 (0,00335)	**0,71314 (0,29015)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
NSGAII_0.01	1	0.01	3.4285	0,72001 (0,02570)	**0,59560 (0,07236)**	**0,70141 (0,15530)**	**0,61785 (0,00274)**	**0,66703 (0,28845)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
NSGAII_0.005	1	0.005	2.4285	0,73652 (0,02579)	**0,67685 (0,11302)**	**0,72136 (0,11490)**	**0,61829 (0,00180)**	**0,76253 (0,25008)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
NSGAII_0.00005	1	0.00005	2.8571	**0,77935 (0,02764)**	**0,68219 (0,09705)**	**0,68787 (0,12214)**	**0,61811 (0,00342)**	**0,65738 (0,25298)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
NSGAII_0.001	1	0.001	2.8571	**0,76397 (0,03029)**	**0,68158 (0,08319)**	**0,70057 (0,13410)**	**0,61889 (0,00202)**	**0,65176 (0,23127)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**

Tabela A.4: Teste de parâmetros para a HITO-NSGAII-MAB

Alg	W	C	Rank	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health Watcher**	**JBoss**
NSGAII	300	2.0	6.0	0,72830 (0,09064)	0,55803 (0,17465)	0,60543 (0,22320)	**0,62890 (0,01673)**	**0,84460 (0,15104)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
NSGAII	150	2.0	2.7142	**0,80732 (0,03949)**	**0,79522 (0,06178)**	**0,80877 (0,10761)**	**0,64069 (0,00516)**	**0,90312 (0,10212)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
NSGAII	150	1.0	3.4285	**0,77511 (0,07778)**	**0,75963 (0,16827)**	**0,82017 (0,09203)**	**0,63712 (0,01061)**	**0,86111 (0,12157)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
NSGAII	300	0.5	6.4285	0,73821 (0,08360)	0,63100 (0,19178)	0,61452 (0,25097)	**0,62629 (0,01848)**	0,75699 (0,15463)	**1,00000 (0,00000)**	**0,96667 (0,18256)**
NSGAII	150	0.5	3.5714	**0,81304 (0,03403)**	**0,76030 (0,09320)**	**0,79556 (0,09100)**	**0,64000 (0,00662)**	0,84890 (0,10243)	**1,00000 (0,00000)**	**1,00000 (0,00000)**
NSGAII	150	5.0	3.2857	0,76141 (0,01656)	0,67172 (0,05246)	**0,80762 (0,06718)**	**0,64073 (0,00384)**	**0,95094 (0,05205)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**
NSGAII	300	1.0	7.1428	0,70250 (0,08509)	0,59447 (0,19397)	0,51347 (0,23738)	**0,62375 (0,01806)**	0,83894 (0,14884)	**0,96667 (0,18256)**	**1,00000 (0,00000)**
NSGAII	300	5.0	3.4285	0,76150 (0,01583)	0,66930 (0,04797)	**0,81936 (0,04961)**	**0,64042 (0,00352)**	**0,91104 (0,09703)**	**1,00000 (0,00000)**	**1,00000 (0,00000)**

Tabela A.5: Teste de parâmetros para a HITO-SPEA2-CF

Alg	α	β	Rank	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health Watcher**	**JBoss**
SPEA2	1	0.01	3.5714	0,77025 (0,02878)	**0,70874 (0,05840)**	**0,52480 (0,13459)**	**0,65100 (0,00649)**	**0,54175 (0,34115)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
SPEA2	1	0.00005	2.0	**0,79744 (0,02880)**	**0,71896 (0,07149)**	**0,59099 (0,15242)**	**0,65188 (0,00520)**	**0,62120 (0,29205)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
SPEA2	1	0.05	3.7142	0,73167 (0,02376)	0,58283 (0,05903)	**0,57712 (0,12692)**	**0,65076 (0,00362)**	**0,54890 (0,34744)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
SPEA2	1	0.005	3.9999	**0,77985 (0,03023)**	**0,69184 (0,10562)**	**0,57695 (0,13529)**	**0,65063 (0,00587)**	**0,46450 (0,31496)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
SPEA2	1	0.10	4.7142	0,72864 (0,02833)	0,54759 (0,07536)	0,45634 (0,15026)	0,64956 (0,00524)	**0,61199 (0,32857)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**
SPEA2	1	0.001	3.0	**0,80443 (0,02460)**	**0,70404 (0,10892)**	**0,57308 (0,13354)**	**0,65241 (0,00297)**	**0,51840 (0,34252)**	**0,25498 (0,00000)**	**0,25498 (0,00000)**

Tabela A.6: Teste de parâmetros para a HITO-SPEA2-MAB

Alg	W	C	Rank	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health Watcher**	**JBoss**
SPEA2	300	2.0	7.0	0,72862 (0,09064)	0,51310 (0,20612)	**0,65042 (0,24049)**	0,71810 (0,01331)	**0,66681 (0,17181)**	**0,96667 (0,18256)**	**0,50495 (0,00000)**
SPEA2	150	1.0	4.2857	**0,75783 (0,10052)**	**0,64694 (0,17542)**	**0,65091 (0,19221)**	**0,72512 (0,01227)**	**0,70398 (0,17561)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**
SPEA2	300	1.0	6.1428	0,70216 (0,08074)	0,55598 (0,19655)	**0,61784 (0,21844)**	**0,72065 (0,01508)**	**0,68161 (0,19638)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**
SPEA2	150	2.0	4.2857	**0,81249 (0,05241)**	**0,62505 (0,14124)**	**0,72530 (0,21188)**	**0,72695 (0,00904)**	**0,72131 (0,17176)**	**1,00000 (0,00000)**	**0,48829 (0,09128)**
SPEA2	150	5.0	3.1428	0,77122 (0,01989)	0,61772 (0,05402)	**0,76982 (0,08155)**	**0,73032 (0,00372)**	**0,75574 (0,15512)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**
SPEA2	150	0.5	2.5714	**0,82625 (0,03245)**	**0,73933 (0,07070)**	**0,77147 (0,10285)**	**0,72773 (0,00755)**	**0,74248 (0,15002)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**
SPEA2	300	0.5	5.8571	0,74150 (0,09443)	0,58301 (0,18330)	**0,61110 (0,21958)**	**0,72187 (0,01190)**	**0,67696 (0,17494)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**
SPEA2	300	5.0	2.7142	0,76976 (0,01910)	0,63334 (0,05020)	**0,78074 (0,07791)**	**0,72964 (0,00326)**	**0,77353 (0,16471)**	**1,00000 (0,00000)**	**0,50495 (0,00000)**

Tabela A.7: Teste de parâmetros para o MOEA/D

Alg	T	Nr	δ	Rank	**MyBatis**	**AJHsqldb**	**AJHotDraw**	**BCEL**	**JHotDraw**	**Health Watcher**	**JBoss**
moead_3	30	3	0.9	0.9999	**0,65038 (0,07552)**	**0,52917 (0,18997)**	**0,58118 (0,16788)**	**0,71937 (0,03907)**	**0,62464 (0,19169)**	**0,85619 (0,25693)**	**0,96094 (0,06666)**
moead_6	30	3	0.9	1.9999	**0,61141 (0,08754)**	**0,47627 (0,17364)**	0,44226 (0,16538)	**0,70599 (0,04427)**	0,48761 (0,19073)	**0,77279 (0,22084)**	**0,89736 (0,22501)**
moead_30	30	30	0.9	2.9999	0,55983 (0,09768)	0,37058 (0,20811)	0,30981 (0,17323)	**0,66963 (0,09428)**	0,34822 (0,14435)	0,66092 (0,29928)	0,69291 (0,34941)