Universidade Federal do Paraná

Ricardo Henrique Remes de Lima

Um estudo sobre configuração automática do algoritmo de Otimização por Enxame de Partículas Multiobjetivo

Curitiba PR
2017

Ricardo Henrique Remes de Lima

Um estudo sobre configuração automática do algoritmo de Otimização por Enxame de Partículas Multiobjetivo

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Aurora Pozo.

Curitiba PR
2017

L732e Lima, Ricardo Henrique Remes de
Um estudo sobre configuração automática do algoritmo de Otimização por Enxame de Partículas Multiobjetivo / Ricardo Henrique Remes de Lima. – Curitiba, 2017.
55 f ; il. color : 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2017.

Orientador: Autora Trinidad Ramirez Pozo
Bibliografia: p. 52-55.

1. Algoritmos. 2. Otimização matemática. 3. Algoritmos genéticos. 4. Programação genética. I. Universidade Federal do Paraná. II.Pozo, Autora Trinidad Ramirez. III. Título.

CDD: 005.1

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós-Graduação INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **RICARDO HENRIQUE REMES DE LIMA** intitulada: **Um estudo sobre configuração automática do algoritmo de Otimização por Enxame de Partículas Multiobjetivo**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO.

Curitiba, 10 de Março de 2017.

AURORA TRINIDAD RAMIREZ POZO
Presidente da Banca Examinadora (UFPR)

PAULO HENRIQUE SIQUEIRA
Avaliador Externo (UFPR)

LUZIA VIDAL DE SOUZA
Avaliador Externo (UFPR)

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

Resumo

O desempenho de algoritmos bio-inspirados está diretamente relacionado a uma escolha adequada de componentes e parâmetros de projeto. Para aumentar a robustez destes métodos e facilitar a sua utilização para usuário comum, pesquisas recentes focam no estudo de estratégias que automaticamente configurem algoritmos. Uma das principais abordagens utilizadas é a Programação Genética (PG), baseada em algoritmos evolutivos, ela evolui uma população de programas de computador através da aplicação de operadores de cruzamento e mutação para resolver o problema em questão. A Evolução Gramatical (GE) é um tipo de PG que utiliza gramáticas livres de contexto para a definição dos componentes do programa. Outra alternativa de configuração automática de algoritmos é a utilização de algoritmos de otimização: diversas ferramentas têm sido desenvolvidas neste contexto, entre elas destacam-se a *Iterated Race* (IRACE), um framework que utiliza conceitos de uma "corrida" entre os candidatos para selecionar as melhores configurações. Nesta dissertação o foco de estudo é a configuração automática de algoritmos e como caso de estudo escolhemos o algoritmo de Otimização por Enxame de Partículas Multiobjetivo (MOPSO). O MOPSO, assim como outros algoritmos de otimização estudados no nosso grupo de pesquisa, possui diversos componentes que podem ser alterados de acordo com a necessidade do usuário e o problema considerado. As duas técnicas Evolução Gramatical e o IRACE serão utilizadas. Experimentos foram realizados para avaliar ambas as técnicas na geração de projetos de MOPSO e verificar se os algoritmos gerados conseguem superar o desempenho de algoritmos refinados manualmente. Os resultados obtidos indicam que é possível gerar projetos MOPSO com desempenho similar e resultados competitivos.

Palavras-chave: evolução gramatical, projeto automático de algoritmos, otimização por enxame de partículas.

Abstract

The performance of bio-inspired algorithms is directly related to an appropriate choice of components and design parameters. To increase the robustness of these methods and simplify their use for ordinary users, recent research focuses on the study of strategies that automatically configure algorithms. One of the main approaches used is Genetic Programming (GP), based on evolutionary algorithms, it evolves a population of computer programs through the application of crossover and mutation operators to solve the problem in question. Grammatical Evolution (GE) is a type of GP that uses context-free grammars to define program components. Another alternative of automatic algorithm configuration is the use of optimization algorithms, several tools have been developed in this context, among them Iterated Race (IRACE), a framework that uses concepts of a "race" among the candidates to select the best settings. In this dissertation the focus of study is the automatic configuration of algorithms and as a case study we chose the Multi-objective Particle Swarm Optimization algorithm (MOPSO). The MOPSO, as well as other optimization algorithms studied in our research group, has several components that can be modified according to the user needs and the problem considered. The two techniques Grammatical Evolution and IRACE will be used. Experiments were performed to evaluate both techniques in the generation of MOPSO designs and to verify if the generated algorithms can outperform manually tunned algorithms. The results indicate that it is possible to generate MOPSO designs with similar performance and competitive results.

Keywords: grammatical evolution, automatic design, particle swarm optimization.

Sumário

Lista de Publicações

Durante o período do Mestrado foram publicados alguns artigos científicos em congressos. Estes trabalhos envolvem tanto pesquisas feitas em disciplinas, que serviram como subsídio para o desenvolvimento da dissertação, quanto pesquisas relacionadas ao tema abordado nesta dissertação. Os trabalhos abaixo listados foram publicados ou estão em processo de publicação:

Publicações em congressos nacionais:

- V. D. da Fontoura, R. H. R. de Lima, A. T. R. Pozo, and R. S. Hermida. Algoritmos Multiobjetivos aplicados ao Problema de Predição de Estruturas Proteínas. In C. J. A. Bastos Filho, A. R. Pozo, and H. S. Lopes, editors, *Anais do 12 Congresso Brasileiro de Inteligência Computacional*, pages 1–6, Curitiba, PR, 2015. ABRICOM

Publicações em congressos internacionais:

- R. H. R. de Lima and A. T. R. Pozo. A Study on Auto-configuration of Multi-objective Particle Swarm Optimization Algorithm. In *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC'17)*. IEEE, 2017. Aceito para publicação

Publicações de capítulo de livro:

- V. D. da Fontoura, R. H. R. de Lima, A. T. R. Pozo, and R. S. Hermida. Multi-objective approach to the Protein Structure Prediction Problem. In *Evolutionary Multi-objective System Design*, 2015. Em produção

Capítulo 1

Introdução

Atualmente o ser humano busca otimizar a realização de suas tarefas diárias. Foi pensando nisso que muitos processos antes realizados manualmente, foram sendo substituídos por máquinas e computadores que realizam o mesmo serviço mais rapidamente e de forma mais eficiente. A ideia de fazer com que os computadores resolvam diversos problemas de forma automática é primordial para a inteligência artificial, aprendizado de máquina e para a vasta área que engloba o que Turing chamou de *inteligência de máquina* [41].

O foco dentro da área de otimização é a resolução de problemas de minimização ou maximização, sendo eles mono ou multiobjetivo, podendo ou não estarem sujeitos a restrições. Várias técnicas de resolução de problemas foram propostas, baseadas nos mais diversos conceitos. Dentre as abordagens existentes, as mais utilizadas são algoritmo de Otimização por Enxame de Partículas (PSO) [26], Algoritmo Genético (GA) [25], Otimização por Colônia de Formigas (ACO) [20], Evolução Diferencial (DE) [37] ou Arrefecimento Simulado (SA) [49].

O PSO é uma meta-heurística desenvolvida nos anos 1990 inspirada nos conceitos envolvidos no comportamento social de revoadas de pássaros procurando por alimentos [40]. O algoritmo tem ganho atenção durante os anos devido a sua flexibilidade e também bons resultados para diferentes aplicações [33]. Posteriormente, o algoritmo ganhou uma extensão para permitir que pudesse ser aplicado a problemas multiobjetivo, surgindo então o *Multi-objective Particle Swarm Optimization* (MOPSO). Foram introduzidos dois novos mecanismos ao PSO, chamados de método de arquivamento e método de seleção de líder, técnicas que auxiliam o algoritmo a manipular e armazenar as melhores soluções levando em consideração os conceitos de dominância de Pareto [12].

Apesar do bom desempenho que estes algoritmos podem obter, o projeto de algoritmos eficientes para a resolução de problemas de otimização, continua sendo uma área desafiadora [13]. Segundo a literatura, a combinação de algoritmos pode obter melhor resultados do que o desempenho individual destes para determinados problemas. Porém, técnicas de combinação, ou automatização desse processo de projeto de novos algoritmos ainda são pouco exploradas [13], sendo a programação genética um tipo de abordagem que tem se destacado nesse sentido.

A Programação Genética (PG) é um método sistemático para criação e modificação automática de programas de computador. A PG é considerada um método independente de domínio, sendo uma extensão de algoritmo evolutivos baseados em população, que evolui um conjunto de programas de computador e os modifica pela aplicação de operadores genéticos para resolver um problema [41]. As principais diferenças da PG com GAs por exemplo, estão no fato de que as estruturas das soluções encontradas na população, que normalmente são vetores, não possuem tamanho fixo. Os candidatos são codificados em árvores sintáticas ao invés de linhas de código, que podem ser executadas como um programa e avaliadas como uma solução [38].

Como alternativa à complexidade envolvida na programação genética com árvores, foi desenvolvida uma abordagem para PG que utiliza gramáticas livres de contexto, chamada Evolução Gramatical ou Gramática Evolutiva (GE) [38]. O diferencial está na utilização de uma gramática BNF (*Backus Naur Form*) onde são definidas as regras de geração dos componentes dos programas. Ao invés de árvores, a GE trabalha com um vetor de inteiros que é mapeado segundo a gramática em uma árvore sintática assim como na PG [45].

Além dos algoritmos evolutivos, existem outras abordagens utilizadas na configuração de algoritmos. Dentre estas abordagens o IRACE se destaca devido a sua aplicação na configuração de algoritmos evolutivos multiobjetivo [30]. O IRACE é um *framework* de configuração automática que considera diferentes componentes algorítmicos como parâmetros categóricos a serem definidos. Nesse sentido, o espaço de busca do IRACE representa as diferentes configurações de componentes e parâmetros para o algoritmo na geração de um projeto único.

1.1 Motivação

Na otimização multiobjetivo os algoritmos costumam possuir mais mecanismos para auxiliá-los na melhora das soluções. No geral tem-se operadores de seleção, cruzamento, mutação, arquivamento de soluções, comparação, reparação, filtragem, classificação, etc. Algoritmos como o PSO e MOPSO podem sofrer com esse tipo de situação, pois para diferentes problemas, diferentes tipos de operadores e técnicas se comportam melhor do que outras. Demonstrando assim a oportunidade para a utilização de uma abordagem que ajude no projeto de algoritmos, ou seja, selecionar os componentes e parâmetros mais adequados de acordo com o problema, ou classe de problemas que se está tentando resolver.

Esta dissertação apresenta um estudo onde explora-se a utilização de diferentes abordagens na configuração automática de algoritmos. As técnicas estudadas foram a GE e o IRACE, ambos métodos promissores. Estes métodos foram aplicados ao algoritmo MOPSO buscando melhorar seu desempenho de acordo com os problemas considerados, apresentando uma alternativa ao refinamento manual de parâmetros.

1.2 Objetivos

Este trabalho permite expandir os estudos sobre o projeto automático de algoritmos, observar a viabilidade da abordagem e identificar os aspectos positivos e negativos.

Dentre os objetivos definidos para este trabalho, o principal é a aplicação da abordagem utilizando os algoritmos GE e IRACE para evoluir projetos do algoritmo MOPSO buscando mostrar os benefícios da configuração automática de algoritmos. Através de experimentos espera-se ser possível gerar MOPSOs que obtenham resultados competitivos em comparação com algoritmos conhecidos.

Os objetivos específicos deste trabalho foram definidos como:

- Estudar a configuração automática de algoritmos e os métodos de Evolução Gramatical e IRACE;
- Elaborar uma gramática para o MOPSO de acordo com componentes e parâmetros do algoritmo;
- Realizar experimentos com a aplicação das técnicas de configuração automática para a geração de algoritmos MOPSO.

- Realizar experimentação com os melhores projetos MOPSO gerados, comparar seus resultados com outra técnica conhecida;

1.3 Estrutura do Trabalho

Esta dissertação está organizada da seguinte forma.

Capítulo 2 apresenta um resumo sobre a área de otimização para então introduzir os algoritmos evolutivos e mais detalhadamente os algoritmos PSO e MOPSO.

Capítulo 3 apresenta os conceitos da programação genética, explicando seu funcionamento, e conceitos de Evolução Gramatical e como essa abordagem é utilizada na configuração automática de algoritmos.

Capítulo 4 explica detalhadamente o IRACE, sua atuação na configuração de algoritmos e seu funcionamento.

Capítulo 5 realiza o levantamento de trabalhos relevantes da área, incluindo aqueles que serviram de inspiração e base para esta dissertação.

Capítulo 6 apresenta a proposta de trabalho, o *framework* desenvolvido, as estruturas, operadores e detalhes necessários para realização dos experimentos.

Capítulo 7 descreve a etapa de experimentação realizada, apresentação dos problemas considerados, análise dos resultados e desempenho da abordagem proposta.

Capítulo 8 realiza um breve resumo, apresentando a conclusão das ideias e propostas para trabalhos futuros.

Capítulo 2

Algoritmos Evolutivos e Otimização por Enxame de Partículas

Este capítulo apresenta os conceitos necessários para compreensão sobre a área de otimização no geral, descrevendo como os algoritmos evolutivos são usados para resolução de problemas, citando um dos principais exemplos, os algoritmos genéticos. Depois, é feita uma introdução ao algoritmo de otimização por enxame de partículas e seus conceitos, população e operadores, que são importantes para entendimento dessa proposta.

2.1 Visão Geral sobre Otimização

Um problema de otimização consiste em encontrar soluções que são ótimas ou quase ótimas considerando alguns objetivos [44]. Cada problema é caracterizado basicamente como sendo a minimização ou maximização de uma função objetivo, podendo ou não estar sujeito a restrições. Eles também podem ser classificados quanto a quantidade de objetivos.

Um problema mono objetivo, é aquele que possui apenas uma função objetivo a ser otimizada. Como por exemplo, encontrar o valor de x para que a função $x^2 + x + 70 = 1000$ fosse válida. Neste caso somente o valor de x deve ser encontrado sem restrições.

Problemas multiobjetivo possuem duas ou mais funções objetivo que normalmente são conflitantes. Um exemplo simples pode ser visto na Equação 2.1, nela o valor de x deve ser encontrado, mas para isso é preciso otimizar ambas as funções $f(x)$ e $g(x)$.

$$\begin{aligned} \text{Min } f(x) &= x^2 \\ \text{Min } g(x) &= (x-1)^2 \end{aligned} \tag{2.1}$$

Otimização Estocástica é uma sub área da Otimização. É uma classe de algoritmos e técnicas de otimização que adicionam um certo grau de aleatoriedade para encontrar soluções ótimas (ou o mais próximo disso) em problemas [32]. Algoritmos Evolutivos (AEs) são as técnicas de otimização estocástica mais comuns utilizados para resolução de problemas de otimização. Dentre essas técnicas podem-se destacar os Algoritmos Genéticos (AGs) que se inspiram nos conceitos da evolução das espécies, propostos por Darwin, para evoluir uma população de soluções, combinando-as e aplicando pequenas alterações, privilegiando as melhores soluções para que continuem no processo de evolução até que um critério de parada seja atingido [34].

Russell Eberhart e James Kennedy propuseram em 1995 um método para otimização de funções não-lineares. Este método, chamado de Otimização por Enxame de Partículas (*Particle*

Swarm Optimization - PSO), tem suas raízes firmadas em duas principais metodologias. A principal delas é a relação com vida artificial (chamada *A-life*) em geral, revoada de pássaros, cardume de peixes, e a teoria de enxames em particular. A segunda, é sua relação com a computação evolutiva, tendo laços com Algoritmos Genéticos e também estratégias evolutivas [21]. Por fim, a ideia de trabalhar com Enxame de Partículas surgiu enquanto se estava analisando como produzir inteligência computacional com a exploração de interação social simples, ao invés de interações puramente individuais de habilidades cognitivas [27].

Segundo Eberhart [21], o PSO compreende conceitos simples, requerendo apenas operadores matemáticos primitivos sendo computacionalmente baratos em termos de memória e velocidade. O algoritmo demonstra ter um bom desempenho em funções normalmente usadas em teste para algoritmos genéticos, mostrando-se uma abordagem promissora para problemas complexos.

Otimização por Enxame de Partículas pode ser utilizada para resolver diversos problemas que Algoritmos Genéticos resolvem. Porém, este método de otimização possui algumas características que os GAs não possuem, como o conceito de interação entre grupos de soluções e também pelo algoritmo PSO possuir memória.

2.2 Dinâmica de Partículas

No PSO a busca por soluções ocorre colocando um número de elementos (partículas) no espaço de busca do problema, onde cada elemento avalia a função de aptidão para sua posição atual. Cada partícula então calcula seu próximo movimento pelo espaço de busca combinando informações sobre sua atual e melhor solução encontrada até então, com informações oferecidas por outras partículas, além da aplicação de algumas perturbações aleatórias. A próxima iteração tem início após todas as partículas terem sido movidas. Eventualmente toda a população acaba se comportando como uma revoada de pássaros coletivamente procurando por comida, movendo-se próximos a uma área ótima da função de aptidão (*fitness*) [40].

Cada partícula é representada por três vetores de N-dimensões, onde N é o número de dimensões do espaço de busca. O vetor $\vec{x}_i$ representa a posição atual, $\vec{p}_i$ a última melhor posição, e $\vec{v}_i$ a velocidade.

A atual posição $\vec{x}_i$ é considerada como um conjunto de coordenadas que descrevem um ponto no espaço. Em cada iteração do algoritmo, a posição atual é avaliada como uma solução do problema. Caso essa posição seja a melhor que qualquer outra encontrada até o momento para aquela partícula, suas coordenadas são salvas no segundo vetor, $\vec{p}_i$, também chamado de $pBest$, que é usado para cálculo da velocidade. O objetivo é continuamente encontrar melhores soluções, atualizando os valores de $\vec{p}_i$. Cada partícula "caminha" pelo espaço adicionando-se a velocidade $\vec{v}_i$ ao vetor de posição $\vec{x}_i$, caracterizando um "passo" [40].

A resolução de problemas através de uma população é um fenômeno emergente do comportamento individual das partículas através de interações. As populações são organizadas segundo uma topologia, tipicamente bidirecional que conecta pares de partículas. Um exemplo de topologia diz que, se a partícula j está na vizinhança de i, então i também está na vizinhança de j. Cada partícula comunica-se com outras partículas, e é afetada pelo melhor ponto encontrado por qualquer membro da sua vizinhança topológica. O melhor ponto é o vetor $\vec{p}_i$ para o melhor vizinho, denotado como $\vec{p}_g$ ou $gBest$. A velocidade adicionada à cada partícula durante o processo de evolução é ajustada para que cada partícula estocasticamente oscile dentre as regiões de $pBest$ e $gBest$. O algoritmo do PSO está representado no Algoritmo 1.

O algoritmo começa inicializando o enxame, para então realizar a avaliação inicial da população e atualização do componente global. Após isso é iniciado um laço, dentro dele os

Algoritmo 1: Otimização por Enxame de Partículas

```
1  inicialização do enxame;
2  avaliação das soluções;
3  atualização do componente individual;
4  atualização do componente global;
5  enquanto não for satisfeito o critério de parada faça
6      para cada partícula ∈ enxame faça
7          atualização da velocidade;
8          atualização da posição;
9          turbulência;
10         avaliação do desempenho;
11         atualização do componente individual;
12         atualização do componente global;
13     fim
14 fim
15 return componente global;
```

seguintes passos são executados para cada partícula da população. É atualizada a velocidade e posição da partícula de acordo com as equações 2.2 e 2.3. Em seguida, dependendo da abordagem, é executado o operador de turbulência (mutação), para depois realizar a avaliação do desempenho e por fim a atualização do $pBest$ e $gBest$. Por fim, se o critério de parada for atingido, o algoritmo termina e a solução retornada é o componente global $gBest$, caracterizada como a melhor solução.

Cada partícula tem sua velocidade para cada dimensão restrita a no máximo um valor $Vmax$. Se a soma das acelerações exceder $Vmax$, que é um parâmetro dado pelo usuário, então a velocidade naquela dimensão é limitada em $Vmax$. O parâmetro $Vmax$ é importante para determinar quais regiões entre a posição atual e a melhor ($gBest$) que são usadas na busca. Se $Vmax$ é um valor muito alto, as partículas podem ignorar boas soluções. Se $Vmax$ é um valor muito pequeno, as partículas podem não explorar as regiões o suficiente, ficando presas em regiões chamadas de ótimos locais [40].

$$\vec{v}_{i+1} = \vec{v}_i + C_1 * r_1 * (\vec{p}_i - \vec{x}_i) + C_2 * r_2 * (\vec{p}_g - \vec{x}_i) \tag{2.2}$$

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_{i+1} \tag{2.3}$$

As variáveis C_1 e C_2 na equação 2.2 determinam o peso dos termos de aceleração que guiam as soluções em direção ao $pbest$ e $gbest$. Normalmente C_1 e C_2 são chamados de coeficientes de aceleração. A influência destes parâmetros na função de aptidão pode ser interpretada como força de atração, podendo levar uma partícula a sofrer maior influência de $pbest$ ou $gbest$. O ajuste deste parâmetro altera a "tensão" do sistema. Valores baixos permitem que partículas passem longe das regiões promissoras antes de serem puxadas de volta, enquanto que valores altos resultam em movimentos mais bruscos em direção a região alvo [40]. Além disso, as variáveis $r1$ e $r2$ são valores aleatoriamente escolhidos entre 0 e 1 que servem para acrescentar um grau de incerteza à busca.

2.3 Operadores

O PSO possui alguns operadores básicos que são utilizados durante seu processo evolutivo. O primeiro deles é o chamado operador de velocidade para realizar a atualização da posição das partículas no espaço. O segundo é o operador de mutação, também conhecido como turbulência, que assim como outros algoritmos evolutivos, realiza uma pequena alteração na solução para tornar possível serem geradas soluções com maior diversidade.

Para o operador de velocidade existem diversas técnicas além da fórmula padrão apresentada (Equação 2.2). Dentre todos os possíveis métodos, podemos citar Peso de Inércia [47], Coeficientes de Constrição [11] e Restrição de Velocidade [36] como exemplos mais relevantes para esta dissertação.

2.3.1 Peso de Inércia

Motivados por um maior controle sobre o escopo da busca, redução da importância do parâmetro $Vmax$, e talvez eliminá-lo, uma modificação na equação de atualização de velocidade foi proposta por Shi e Eberhart em [47], dando origem a Equação 2.4:

$$\vec{v}_{i+1} = w * \vec{v}_i + C_1 * r_1 * (\vec{p}_i - \vec{x}_i) + C_2 * r_2 * (\vec{p}_g - \vec{x}_i) \tag{2.4}$$

A equação é idêntica a Equação 2.2, sendo apenas adicionado um fator w multiplicando $\vec{v}_i$, onde w é um termo que representa o "peso da inércia". A escolha de bons valores para w provocam um balanço entre exploração local e global, resultando em menos iterações em média para encontrar uma solução que seja suficiente, porém, sem garantias de ser ótima [47].

Em experimentos realizados por Eberhart e Kennedy em [22], foi constatado que nem sempre $Vmax$ pode ser descartado, mas elimina a necessidade de pensar em como ajustar $Vmax$ a cada vez que o algoritmo é usado.

2.3.2 Coeficientes de Constrição

Um fator de constrição foi proposto para o PSO originalmente por Clerc e Kennedy em [11], como sendo necessário para assegurar convergência do algoritmo. A Equação 2.5 mostra de forma simplificada como esse fator é aplicado ao PSO, onde K é uma função de c_1 e c_2 mostrado na Equação 2.6.

$$\vec{v}_{i+1} = K * [\vec{v}_i + C_1 * r1 * (\vec{p}_i - \vec{x}_i) + C_2 * r2 * (\vec{p}_g - \vec{x}_i)] \tag{2.5}$$

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{2.6}$$

onde

$$\begin{cases} \varphi = C_1 + C_2 & \text{se } C_1 + C_2 > 4 \\ K = 1 & \text{se } C_1 + C_2 \leq 4 \end{cases} \tag{2.7}$$

Quando $C_1 + C_2 > 4$, K é calculado pela equação 2.6, porém, nos demais casos onde $C_1 + C_2 \leq 4$, o valor de K é 1 e a equação 2.6 não é usada, fazendo com que a velocidade da partícula não seja afetada.

Partículas que utilizaram do fator de constrição convergem sem a necessidade da utilização de um parâmetro V_{max}. O procedimento sugerido nestes casos é definir V_{max} igual a

X_{max}, que representa o alcance de cada variável em cada dimensão. O resultado é um algoritmo PSO não dependente de parâmetros específicos do problema. Considerado o algoritmo canônico atualmente [47].

2.3.3 Restrição de Velocidade

O operador de velocidade chamado Restrição de Velocidade (*Speed Constraint*) é utilizado no algoritmo SMPSO [36], que funciona unindo as equações de Peso de Inércia e Coeficientes de Constrição, como uma forma de controlar a velocidade das partículas sem o uso de limitantes.

Ao invés de usar limitantes superior e inferior para controlar a velocidade das partículas, o coeficiente de constrição (Equação 2.6) foi adotado, obtendo o fator K proposto em [11].

Adicionalmente, foi introduzido um mecanismo onde a velocidade acumulada para cada j (em cada partícula) é limitado por meio da função de restrição de velocidade:

$$v_{i,j}(t) = \begin{cases} delta_j & \text{se } v_{i,j}(t) > delta_j \\ -delta_j & \text{se } v_{i,j}(t) \leq delta_j \\ v_{i,j}(t) & \text{caso contrário} \end{cases} \tag{2.8}$$

onde

$$delta_j = \frac{(limite\_sup_j - limite\_inf_j)}{2} \tag{2.9}$$

Resumindo o processo, a velocidade das partículas é calculada de acordo com a equação do Peso de Inércia (Equação 2.4); a velocidade resultante é então multiplicada pelo fator de constrição (Equação 2.6) e o valor resultante é limitado usando a Equação 2.8.

2.4 MOPSO

A otimização multiobjetivo é uma parte integral da área de otimização e possui grande importância prática, pelo fato de que muitos dos problemas do mundo real são modelados como tendo objetivos conflitantes [17]. Ao longo do crescimento da área de otimização, o uso e desenvolvimento de técnicas multiobjetivo baseadas em heurísticas cresceu muito. Objetivando a produção de algoritmos mais eficientes, mecanismos inteligentes para manter diversidade e o uso de populações menores foram sendo criados.

Problemas com múltiplos objetivos são caracterizados por não haver uma única solução considerada ótima para solucionar o problema, e sim um conjunto de soluções que satisfazem os objetivos em proporções diferentes. Esse conjunto de soluções é chamado de conjunto ótimo ou conjunto de Pareto. Duas soluções distintas devem ser avaliadas levando em consideração uma relação de dominância, chamada de Dominância de Pareto.

2.4.1 Dominância de Pareto

Segundo a definição apresentada por Alvarez-Benitez em [1], dentro da otimização multiobjetivo deseja-se alcançar as melhores soluções para D objetivos: $y_i = f_i(x)$, onde $i = 1, ..., D$ e onde cada objetivo depende de um vetor x de K parâmetros ou variáveis de decisão. Estes parâmetros também podem estar sujeitos a J constantes: $e_j(x) \geq 0$ para $j = 1, ..., J$, que são restrições do problema.

Sem perda de generalidade pode-se assumir que estes objetivos são de minimização. Dessa forma o problema pode ser representado como nas Equações abaixo:

$$\text{minimizar } \mathbf{y} = \mathbf{f(x)} \equiv (f_1(x), f_2(x), ..., f_D(x)) \tag{2.10}$$

$$\text{sujeito a } \mathbf{e(x)} \equiv (e_1(x), e_2(x), ..., e_J(x)) \geq 0 \tag{2.11}$$

Um vetor de decisão $\mathbf{u}$ é dito que domina estritamente outro $\mathbf{v}$ (denotado $\mathbf{u} \prec \mathbf{v}$) se $f_i(\mathbf{u}) \leq f_i(\mathbf{v})\ \forall_i = 1, ..., D$ e $f_i(\mathbf{u}) < f_i(\mathbf{v})$ para algum i; ou de forma menos rigorosa $\mathbf{u}$ domina fracamente $\mathbf{v}$ (denotado $\mathbf{u} \preceq \mathbf{v}$) se $f_i(\mathbf{u}) \leq f_i(\mathbf{v})$ para todo i. Um conjunto de vetores de decisão é dito como sendo um conjunto não-dominado se seus membros não forem dominados por qualquer outro membro. É chamado de Conjunto de Pareto (*Pareto Set*), o conjunto de soluções não dominadas no espaço de decisão. A chamada Fronteira de Pareto (*Pareto Front*) é a imagem do Conjunto de Pareto no espaço de objetivos.

2.4.2 Algoritmo MOPSO

O algoritmo proposto por Coello [12] chamado *Multi-objective Particle Swarm Optimization* (MOPSO) é uma abordagem que estende o PSO e o torna capaz de lidar com problemas multiobjetivo. Para isso, foi proposto para o algoritmo a utilização de um mecanismo de memória externa (chamado repositório) e um mecanismo geográfico para manter a diversidade.

As principais diferenças estão presentes na forma como o algoritmo lida com um conjunto de soluções, utilizando o conceito de dominância, descrito na Seção 2.4.1. Isso implica que não há mais apenas uma só solução considerada a melhor globalmente entre as partículas ($gBest$), mas sim um conjunto de soluções que não são dominadas. O processo completo é apresentado no Algoritmo 2.

Algoritmo 2: Otimização por Enxame de Partículas Multiobjetivo

```
1  inicialização do enxame;
2  inicialização do repositório;
3  avaliação do repositório;
4  enquanto não for satisfeito o critério de parada faça
5      para cada particula ∈ enxame faça
6          seleção de líder;
7          atualização da velocidade;
8          atualização da posição;
9          turbulência;
10         avaliação do desempenho;
11         atualização do componente individual;
12     fim
13     atualização do repositório;
14 fim
15 return repositório;
```

O algoritmo começa inicializando o enxame e o repositório, em seguida é feita a avaliação inicial do repositório. Após isso é iniciado um laço, dentro dele os seguintes passos são executados para cada partícula da população: é realizada a seleção de líder, a melhor solução

global, dentre as soluções do repositório. A partir disso é atualizada a velocidade e posição da partícula. Em seguida, dependendo da abordagem, é executado o operador de turbulência, para depois realizar a avaliação do desempenho e por fim a atualização do $pBest$ se for o caso. Depois, o repositório é atualizado, podendo ocorrer de duas formas: após a atualização de cada partícula; ou após todas as partículas terem sido atualizadas. Por fim, se o critério de parada for atingido, o algoritmo termina e o repositório é retornado como conjunto final de soluções. As linhas em destaque (linhas 6 e 13), estão representando as principais adições com relação ao PSO mono objetivo.

2.4.3 Operadores de Arquivamento e Seleção de Líder

Os operadores do MOPSO funcionam de forma semelhante ao PSO, também utiliza-se um método de atualização da velocidade das partículas e em alguns casos a mutação (turbulência), porém a diferença está no uso do método de arquivamento e da seleção de líder, sendo responsáveis por auxiliar o algoritmo a lidar com várias soluções.

2.4.4 Arquivamento

Para controlar de forma mais eficiente o conjunto de soluções consideradas quase ótimas, um mecanismo chamado repositório é utilizado. Este repositório é definido usualmente com um tamanho fixo, para ir armazenando soluções não-dominadas encontradas durante todo o processo evolutivo. Enquanto o tamanho máximo não for excedido, novas soluções são adicionadas sem restrição, porém, sempre que houver a adição de uma nova solução que exceda o tamanho máximo do repositório, é utilizado algum critério que seleciona qual solução será removida para a adição da nova [12]. Alguns exemplos de técnicas de arquivamento são *Crowding Distance* [18], *Ideal* [5] e *Multilevel Grid Archiving* [29].

- O método de arquivamento de *Crowding Distance* (CD) ilustrado na Figura 2.1, ordena soluções em ordem crescente com base nas funções objetivo. Para cada solução, um valor é atribuído igual a distância entre duas soluções adjacentes, sendo uma maior e a outra menor que a solução avaliada. Para as soluções extremas do conjunto – soluções que possuem um valor com grande diferença para um objetivo com relação aos demais, posicionando-a nos extremos no espaço de busca –, a distância atribuída é infinito. As distâncias são calculadas para cada objetivo e acumuladas para gerar o valor chamado de *crowding distance*. No contexto de arquivamento, quando uma solução deve ser removida para a adição de uma nova, a solução a ser removida será escolhida com base no menor valor de *crowding distance* [24].

- O arquivamento *Ideal* representado na Figura 2.2 consiste em calcular para cada solução no repositório um "ponto ideal". Esse ponto é calculado a partir do melhor valor encontrado para cada objetivo, depois, é calculada a distância de cada solução até o ponto ideal. Quando uma solução precisa ser removida do repositório, aquela com a maior distância do ponto ideal é removida [8].

- O *Multilevel Grid Archiving* (MGA) representado na Figura 2.3, divide o espaço de objetivos em seções. Cada seção levando em consideração a relação de dominância entre elas. Quando uma solução do repositório deve ser removida, é verificado se a nova solução está disposta em uma seção não-dominada. Caso esteja, ela é adicionada enquanto uma solução pertencendo a uma seção dominada é escolhida aleatoriamente para sair. Caso

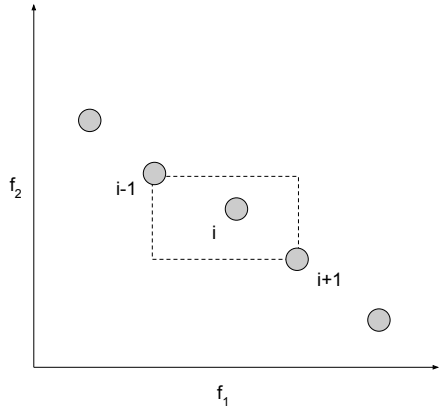

Figura 2.1: Exemplo de técnica de *Crowding Distance* aplicada a solução i, utilizando as soluções $i-1$ e $i+1$ para calcular a distância entre soluções.

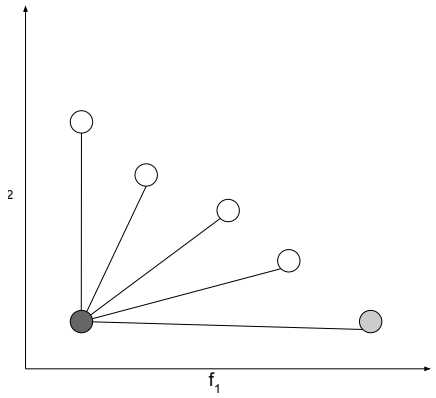

Figura 2.2: Exemplo de aplicação do arquivamento *Ideal*, onde a solução marcada em tom mais claro é a que possui maior distância com relação ao ponto ideal calculado, marcado com tom mais escuro.

a nova solução esteja em uma seção dominada, ela não é adicionada, e caso o espaço de seções não revelar dominância entre elas, o espaço é dividido novamente em seções menores, até que uma seção seja dominada [8].

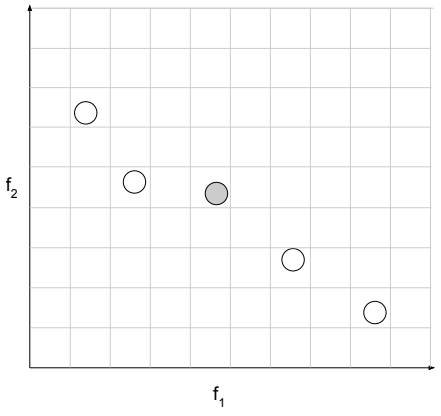

Figura 2.3: Exemplo de aplicação do arquivamento MGA, onde a solução marcada será removida para a adição da solução preenchida.

Para os três métodos apresentados, cada um possui características diferentes quanto ao impacto que eles causam no espaço de soluções. O método *Ideal* por exemplo, prioriza convergência das soluções, enquanto os métodos de *Crowding Distance* e MGA priorizam diversidade.

2.4.5 Seleção de Líder

Na atualização da velocidade das partículas, o componente global $gBest$ funciona de forma diferente quando se trabalha com vários objetivos. Nesse caso é necessário a utilização de um mecanismo para seleção do líder, uma dentre o conjunto de soluções do repositório que será o $gbest$ escolhido a cada iteração, com base em algum critério. Algumas técnicas que podem ser utilizadas são *Crowding Distance* [18], NWSUM [39] e *Sigma* [35].

- Para a seleção de líder o método de *Crowding Distance* (CD) é utilizado para mensurar diversidade, caracterizado pelo maior valor de "*Crowding Distance*" possível que encapsule uma solução sem incluir nenhuma outra. Para determinar um líder são escolhidas duas soluções do repositório, para então escolher aquela com o maior valor de CD.

- O método NWSUM realiza uma atribuição de pesos para as características das partículas, onde aquelas que mais influenciam positivamente uma solução são as que ganham maior peso. A seleção se dá pela soma ponderada dos valores dos objetivos para determinar o líder [39]. A solução escolhida é aquela que possui o maior valor na soma. Este método é utilizado para manter boa diversidade de soluções.

- O método *Sigma* representado na Figura 2.4, funciona calculando a cada iteração os vetores *Sigma* para cada uma das soluções do repositório. Um vetor *Sigma* parte da origem do espaço de objetivos e cruza o ponto que está sendo avaliado naquela iteração. O líder é determinado pela solução do repositório que possui o vetor *Sigma* mais próximo do ponto avaliado.

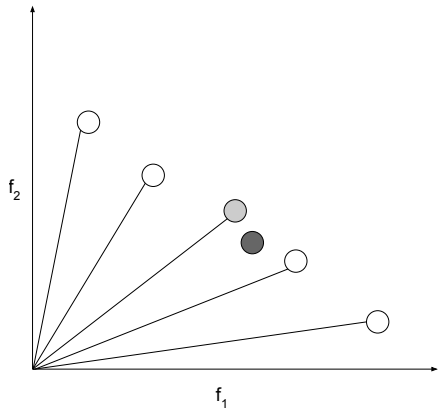

Figura 2.4: Método Sigma sendo aplicado para a solução destacada em tom escuro. A solução marcada em tom claro é a escolhida como líder.

Da mesma forma como os métodos de arquivamento, as técnicas de seleção de líder possuem diferentes características e apresentam desempenhos diferentes entre si. Trabalhos anteriores como Castro e Pozo em [8], mostram que uma boa escolha para os métodos de arquivamento e seleção de líder podem provocar grande impacto e melhoram consideravelmente os resultados da busca.

Capítulo 3

Programação Genética

A Programação Genética (PG) é um processo sistemático para que computadores resolvam problemas de forma automatizada [41]. Com funcionamento semelhante a um Algoritmo Genético, a PG iterativamente evolui uma população de soluções aplicando operadores de seleção, cruzamento e mutação, buscando melhorar a aptidão das soluções e dando preferência às melhores soluções para que façam parte das próximas gerações.

Diferente de um AG, as soluções com que a PG trabalha são programas de computador, e por isso a avaliação das soluções é custosa e toma muito tempo. Tendo isso em mente, viu-se necessário que uma estrutura de representação diferente para os indivíduos da população fosse utilizada [41]. Dentre as várias abordagens para a representação dos indivíduos, as principais são a representação por árvore sintática e gramática [7].

Os passos básicos na PG estão representados no Algoritmo 3. É inicializada uma população de programas, e enquanto não for satisfeito um critério de parada, que pode ser quantidade de gerações, avaliações ou valor de aptidão, cada solução é executada e tem sua aptidão avaliada. Após isso, é feita uma seleção de indivíduos para aplicar os operadores de cruzamento e mutação, que darão origem a novos programas que substituirão os piores na população.

Na PG assim como outros algoritmos evolutivos, se está interessado em quão boas são as soluções que estão sendo geradas. Cada programa é executado e seu comportamento é avaliado. Essa avaliação é quantificada para caracterizar a aptidão daquela solução. Os melhores programas são selecionados para a criação de novos programas para a próxima geração por meio de operadores de Cruzamento e Mutação. O operador de Cruzamento tem como função a criação de um novo programa a partir da combinação de partes aleatórias de dois programas selecionados. A mutação tem como função a criação de um novo programa pela aplicação de uma alteração aleatória aplicada a um programa selecionado [43].

3.1 Representação do Problema

A forma mais comum de PG é baseada na utilização de uma árvore sintática como estrutura de representação para os programas. A árvore é normalmente formada por nós, que são representados por funções básicas, variáveis, constantes ou operações da CPU. A execução destes programas é feita pela passagem de parâmetros para as funções (nós) e o retorno destas servindo de entradas para outras funções, até atingir a raiz da árvore que produz a saída final. Considere a função matemática $sin(cos(x - sinx) + x\sqrt{x})$. Através da Figura 3.1 é possível verificar a árvore sintática que representa essa função. Estes nós precisam fazer parte de um mesmo contexto, caso contrário não seria possível utilizá-los em conjunto. Por exemplo, 10 +

Algoritmo 3: PG Básica

```
1  início
2      inicializar população;
3      enquanto não for satisfeito o critério de parada faça
4          para cada solução faça
5              executa solução;
6              avaliação do desempenho;
7              seleção;
8              cruzamento;
9              mutação;
10         fim
11     fim
12     retorna Melhor solução até então.
13 fim
```

obter_cor() não faz nenhum sentido, a menos que *obter_cor()* retorne um número. Seguindo a mesma ideia, os nós podem possuir certas restrições quanto a ter um certo número de nós-filho: por exemplo, se um nó executa a operação de multiplicação de matrizes, ele pode estar esperando exatamente dois filhos, que representam as matrizes que serão multiplicadas. Por esse motivo, as operações de inicialização e perturbação da GP em árvores estão preocupadas em manter a validade dos indivíduos gerados ao longo do processo [32]. Essa é uma propriedade chamada "fechamento" (*closure*), que diz que qualquer função que recebe um valor, precisa ser capaz de lidar com qualquer valor que pode ser oferecido [23].

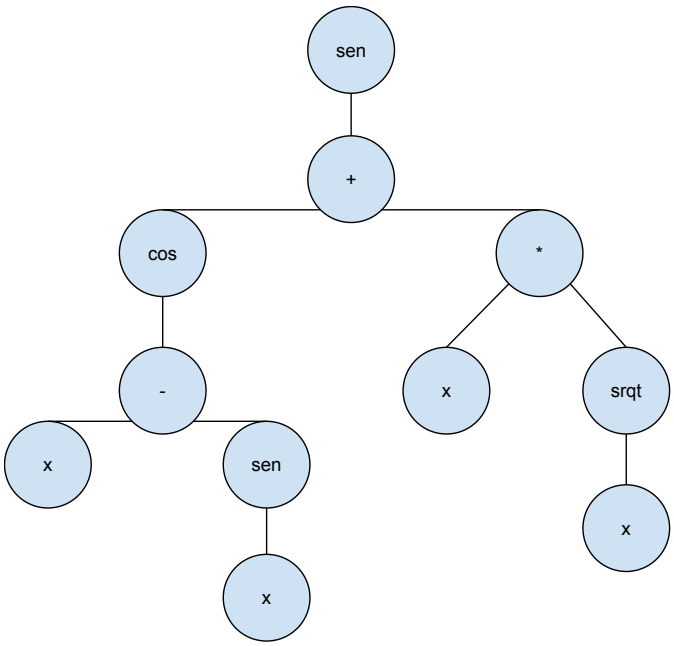

Figura 3.1: Uma árvore que representa uma função matemática. Adaptado de [32].

O cálculo da aptidão para uma árvore igual ao do exemplo dado acima, pode ser por exemplo, tentar aproximar o máximo possível, o resultado da árvore do resultado esperado para um dado conjunto de valores. Supondo para um conjunto na forma $(x_i, f(x_i))$, para $i = 1, ..., 20$, é calculado o erro para cada x que é, $\delta_i = (v_i - f(x_i))^2$, onde para cada par i tem-se o valor de

entrada x e o valor obtido v com a execução da árvore. A aptidão é então calculada como sendo a raiz quadrada do erro total: $\sqrt{\delta_1 + \delta_2 + ... + \delta_{20}}$.

Programas evoluídos pela GP nem sempre serão funções matemáticas. Pode haver casos onde a árvore representa um algoritmo que executa alguns passos, não somente retornar valores, como por exemplo mover uma formiga por um tabuleiro em busca de comida. A Figura 3.2 mostra um exemplo de árvore gerada para guiar a formiga [32].

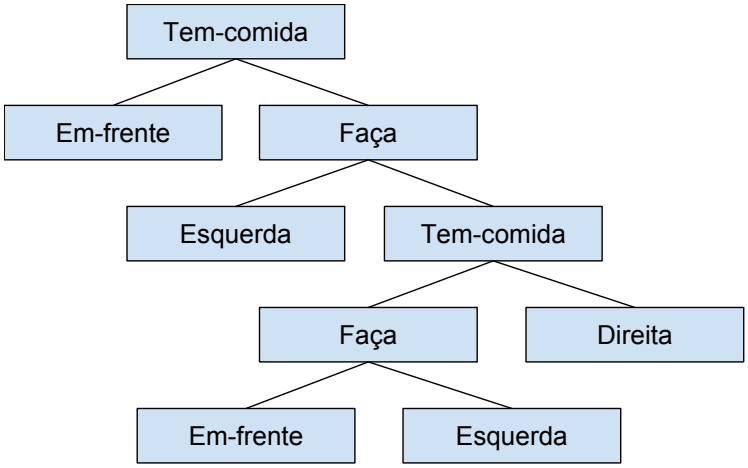

Figura 3.2: Uma árvore para uma formiga artificial. Adaptado de [32].

Para esse exemplo, a operação "*tem-comida*" possui dois filhos. O primeiro é avaliado se a função retornar positivo, caso contrário o segundo é avaliado. A operação "*faça*" também possui dois filhos, porém, neste caso os dois filhos são avaliados na sequência que foram passados.

No geral, a representação vai depender muito do problema que está sendo tratado [32].

3.2 Inicialização

Para o processo de inicialização, as árvores são construídas selecionando os nós de um conjunto de funções repetidamente. Para o exemplo da função matemática, o conjunto poderia ser descrito como as funções: +, -, *, /, sin, cos, $sqrt$, x, ou outras funções. Para cada função existe uma propriedade chamada *arity*, que significa a quantidade de filhos que cada nó precisa ter. Por exemplo, a função sin precisa de apenas um nó filho, a função + precisa de dois filhos, enquanto que x não precisa de nenhum filho. Nós que não possuem filhos *arity* 0, são chamados nós-folha. Os algoritmos levam essas convenções em consideração para construir árvores válidas.

O método mais comum conhecido para geração de árvores é representada pelo Algoritmo 4, ele constrói uma árvore com nós aleatórios até uma profundidade definida:

Algoritmo 4: Grow

```
1 max ← profundidade máxima
2 funções ← conjunto de funções

3 return DoGrow(1, max, funções)
```

Existe uma variante do algoritmo ***Grow*** chamado ***Full*** que constrói árvores completas, dada a profundidade. A única diferença do *Grow* para o *Full* é uma mudança na linha 6, que ao invés de selecionar um nó aleatório, seleciona apenas nós não-folha do conjunto de funções.

Algoritmo 5: DoGrow

```
Entrada: profundidade, max, funções
1  início
2      se profundidade >= max então
3          return Copia(nó-folha aleatório do conjunto funções)
4      fim
5      senão
6          n ← Copia(nó aleatório do conjunto funções)
7          l ← número de filhos esperados pelo nó n
8          para i de 1 até l faça
9              Filho i de n ← DoGrow(profundidade+1, max, funções)
10         fim
11         return n
12     fim
13 fim
```

Na prática cada algoritmo é utilizado metade do tempo, com uma profundidade máxima variada. Esse processo é chamado *Ramped Half-and-Half*. O problema com estes algoritmos é que eles não provêm controle sobre o tamanho da árvore, e tendem a realizar uma distribuição estranha. Para estes casos, Sean Luke em [32] propôs um algoritmo chamado PTC2, que produz uma árvore com o tamanho desejado, ou até esse valor. Seu funcionamento se resume pela expansão da árvore com nós não-folha até que essa quantidade somada com os espaços remanescentes seja igual ou maior ao tamanho desejado, para então completar estes espaços com nós-folha.

3.3 Operadores Genéticos

Os operadores genéticos de Cruzamento e Mutação na PG são diferentes do que usualmente é utilizado em outros algoritmos evolutivos, pois as operações são executadas na estrutura de árvores. Ambas as operações devem ser capazes de serem aplicadas a qualquer nó da estrutura e sempre produzir um resultado válido.

O cruzamento em PG é feito realizando uma recombinação de subárvores. A ideia é simples: os indivíduos pais selecionados para cruzamento tem uma subárvore escolhida aleatoriamente (podendo a raiz também ser escolhida), sendo mais comum buscar pontos semelhantes, para então trocar as subárvores entre os pais gerando assim dois filhos.

No exemplo mostrado pela Figura 3.3, para os dois pais selecionados, é escolhido o ponto que sofrerá o cruzamento, para então realizar a troca das subárvores dos pontos selecionados entre os pais, dando origem aos filhos representados na Figura 3.4.

Por padrão em PG não se realiza mutação com frequência, pois o operador de cruzamento é dito não-homólogo e altamente mutativo [32] dentro da PG. Porém, existem várias formas de realizar mutação, alguns exemplos de métodos de mutação são:

- Escolher uma subárvore de um indivíduo e então substituí-la por uma nova subárvore gerada aleatoriamente;
- Substituir um nó não-folha aleatório por uma de suas subárvores;
- Escolher um nó não-folha e trocar suas subárvores.

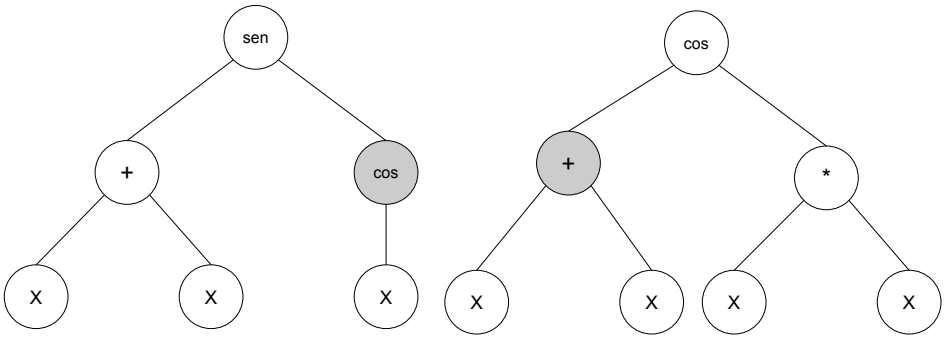

Figura 3.3: Escolha do ponto de cruzamento.

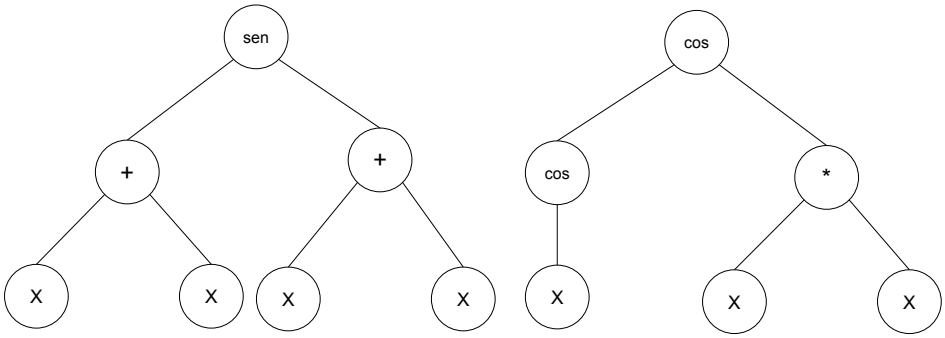

Figura 3.4: Filhos gerados pelo cruzamento.

A Figura 3.5 exemplifica a mutação onde o indivíduo selecionado tem um nó escolhido para ser substituído por uma nova subárvore gerada aleatoriamente.

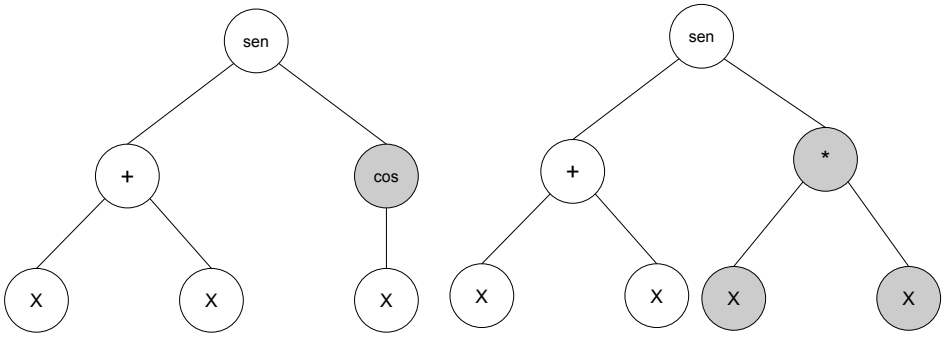

Figura 3.5: Exemplo de mutação em uma árvore sintática.

3.4 Evolução Gramatical

A Evolução Gramatical (*Grammatical Evolution* - GE) é uma abordagem dentro da Programação Genética que propõe o uso de gramáticas livres de contexto para representar os programas ao invés de árvores sintáticas. Diferente da PG, a GE não realiza o processo evolutivo nos programas, mas sim em indivíduos representados por vetores de tamanho variável ou strings binárias que podem ser convertidas em vetores de inteiros. Para isso, um processo de mapeamento é aplicado para traduzir um indivíduo de vetor em um programa por meio de uma gramática BNF [38].

A criação de abordagens que usam gramáticas na PG surgiram principalmente com o objetivo de superar o problema de fechamento, propriedade que descreve que para a PG funcionar, é necessário que os indivíduos, sejam eles árvores sintáticas ou outra estrutura dependendo da abordagem, possam ser manipulados e misturados de forma que o resultado gerado seja sempre um programa válido [38].

Existem três componentes essenciais para um algoritmo de evolução gramatical: 1) a gramática; 2) função de mapeamento; e 3) mecanismo de busca. A utilização de uma gramática para geração de sentenças de uma linguagem arbitrária torna possível a restrição da linguagem enquanto permite a criação de infinitas variações através da aplicação recursiva das regras [7]. A função de mapeamento é o procedimento que permite transformar uma solução de genótipo, que representa os genes (vetor de inteiros) das soluções, para fenótipo, que representa o algoritmo gerado, através da gramática, para que ela seja executada e avaliada. O mecanismo de busca é responsável por manipular e buscar por melhores soluções baseado no resultado da avaliação das existentes.

3.4.1 Gramática BNF

A Gramática é um conjunto estruturado de regras que definem a composição de sentenças para uma dada linguagem. As regras gramaticais proveem uma forma concisa para se restringir uma linguagem enquanto, dependendo da gramática, permite a criação de infinitas variações através da aplicação recursiva das regras [7]. A Gramática é representada pela *Backus-Naur Form* (BNF), uma metassintaxe usada para expressar gramáticas livres de contexto.

Uma gramática consiste basicamente de *terminais*, os quais são itens que aparecem na linguagem (+, -, etc), e *não-terminais*, os quais podem ser expandidos em terminais e não-terminais. A gramática pode ser representada pela tupla (N, T, P, S), onde N é o conjunto de não-terminais, T o conjunto de terminais, P o conjunto de regras de produção que gera sequências de T a partir de N, e S é o símbolo inicial que é membro de N. Quando há uma regra N que possui mais de uma opção possível, as escolhas são separadas pelo símbolo "|" (pipe).

⟨*expr*⟩ ::= ⟨*op*⟩ ⟨*expr*⟩ ⟨*expr*⟩ | ⟨*var*⟩

⟨*op*⟩ ::= + | - | * | /

⟨*var*⟩ ::= x | y | z

Figura 3.6: Exemplo de gramática para geração de equações matemáticas. Adaptado de [7]

A Figura 3.6 mostra uma gramática básica no formado BNF, que pode ser usada para gerar equações matemáticas simples. Nela, pode-se definir os grupos de nós existentes. No grupo de nós terminais, tem-se: $+$, $-$, $*$, $/$, como operações, x, y e z como variáveis. Para o grupo de nós não terminais, tem-se: <op>, <expr> and <var>.

3.4.2 Representação do Problema

A principal diferença entre GE e PG, é que em PG, tanto o genótipo quanto o fenótipo de um indivíduo que está sendo evoluído, são árvores sintáticas, enquanto que em GE, o genótipo e fenótipo possuem estruturas diferentes. O genótipo é representado por uma cadeia linear de *códons* (sequencia de 8 bits, ou posição do vetor de inteiros) de tamanho variável, que então

é decodificado para um vetor de inteiros. O fenótipo, que é a estrutura que define a aptidão, é representado por uma árvore de sintaxe (resultante da aplicação da gramática) [10]. A clara distinção entre genótipo e fenótipo em GE permite que o processo evolutivo ocorra no espaço de busca (genótipo) sem a necessidade de um operador que gere diversidade para o fenótipo [46].

Os programas são gerados a partir de um vetor de inteiros, onde cada posição do vetor possui um valor que é usado para determinar qual regra de produção da gramática BNF será usada, e consequentemente formar a estrutura do programa. Para fins de controle, os valores possíveis para esse vetor podem ser limitados, sendo comum encontrar limites definidos entre 0 e 255 [6]. A partir deste vetor é possível mapeá-lo para gerar o algoritmo (programa) com as funções escolhidas na gramática. O algoritmo gerado é utilizado para o cálculo de aptidão, a Figura 3.7 representa o esquema de forma simplificada.

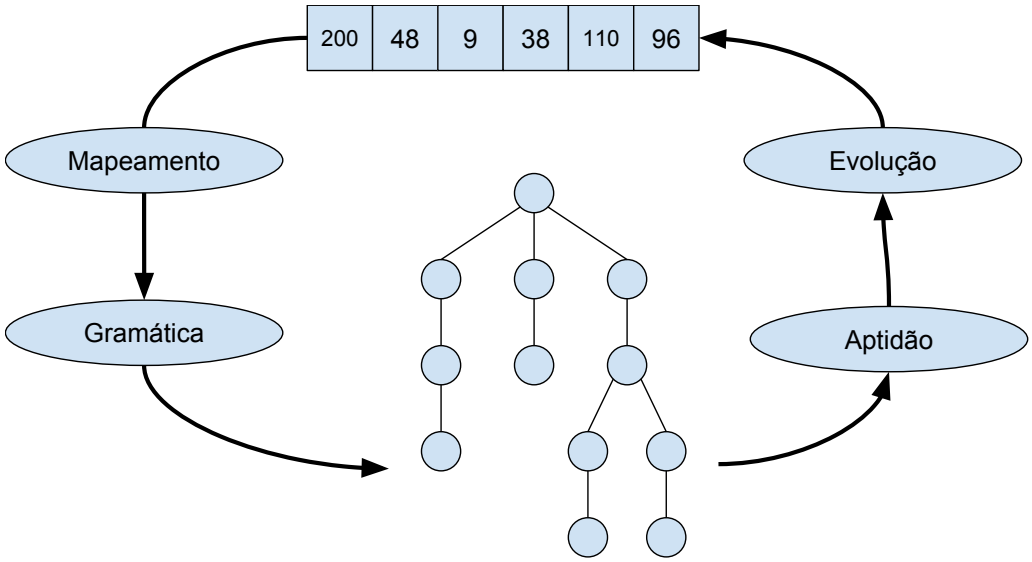

Figura 3.7: Esquema básico utilizado na Evolução Gramatical. Adaptado de [10].

Para o mapeamento genótipo-fenótipo, o seguinte procedimento é feito a partir dos valores inteiros presentes no indivíduo:

$$GR_i = c \ \ mod \ \ nr_i \tag{3.1}$$

onde GR_i é o índice da regra da gramática que será escolhida para o nó não-terminal i, c é o valor do códon do indivíduo, e nr_i é a quantidade de regras possíveis para o não terminal i. A operação *mod* retorna o resto da divisão entre dois números. Há dois casos especiais, quando há mais ou menos genes do que o necessário. No primeiro caso, o mapeamento é feito somente até o gene que ele precisa para gerar um programa válido, os demais não são utilizados. No segundo caso, a solução é tratada de forma cíclica, ou seja, quando não há mais genes disponíveis, o mapeamento começa a utilizar os genes do início até conseguir gerar um programa válido.

Dessa forma, utilizando a gramática apresentada na Figura 3.6, é possível transformar um vetor como por exemplo [0, 9, 3, 7, 11, 4] na expressão $- \ y \ z$ (que está na notação pré-fixa e equivale a $y - z$). Para esse exemplo o processo ocorreu da seguinte forma, seguindo a regra da substituição mais a esquerda:

- $0 \ mod \ 2 = 0$. Substitui a regra inicial $< expr >$ por $< op >< expr >< expr >$;
- $9 \ mod \ 4 = 1$. Substitui a regra $< op >$ pela operação − (subtração);

- 3 *mod* 2 = 1. Substitui a próxima regra mais a esquerda (< *expr* >) por < *var* >;
- 7 *mod* 3 = 1. Substitui o < *var* > pela variável *y*;
- 11 *mod* 2 = 1. Substitui o segundo < *expr* > pela regra < *var* >;
- 4 *mod* 3 = 2. Substitui o < *var* > pela variável *z*.

A diferença do exemplo mostrado para o mapeamento de uma expressão matemática para um programa de computador por exemplo, seria substituir os operadores e variáveis por funções que seriam utilizadas, podendo inclusive incluir na gramática parâmetros e possíveis valores que eles podem assumir.

É possível que dois vetores diferentes gerem o mesmo programa, pois, o mapeamento utiliza a operação módulo para selecionar as regras na gramática. Dois valores podem resultar no mesmo valor para o módulo, como por exemplo 15 MOD 4, e 23 MOD 4 resultam em 3.

3.5 Projeto Automático de Algoritmos com PG

Pode ser observado na literatura, que para alguns problemas, a combinação de algoritmos tem melhor desempenho comparado ao desempenho individual destes algoritmos. O estudo que envolve a combinação de métodos existentes, ou criação de novos algoritmos de forma automática, tem ganho atenção nos últimos anos [13].

O Design Automático de Algoritmos se define basicamente pelo estudo da geração automática de algoritmos com o objetivo de obter um desempenho satisfatório comparado aos algoritmos refinados manualmente [13]. O design de um algoritmo para resolver um problema de otimização é uma tarefa intelectual complexa, e o especialista que está encarregado dessa tarefa deve possuir grande conhecimento dos fundamentos associados a esse domínio [13]. A geração automática de algoritmos tem a capacidade de originar novos procedimentos capazes de utilizar as melhores características de heurísticas e meta-heurísticas já existentes, sugerindo alternativas para resolver problemas difíceis [13]. Claramente não é possível evoluir um algoritmo automaticamente para que ele seja melhor que todos os outros, porém, é possível encontrar um algoritmo que tenha bom desempenho em determinada classe de problemas de interesse [42].

Para que a construção dos algoritmos seja um processo eficiente, é necessário muito cuidado para definir os componentes que podem ser utilizados. Um algoritmo não precisa ser construído totalmente do zero para que haja diferença visível. Os componentes que forem definidos devem ser capazes de causar grande impacto no algoritmo dependendo de quais são utilizados [13].

Péricles Miranda e Ricardo Prudêncio [33] desenvolveram um trabalho que teve muita influência no desenvolvimento das ideias para esta proposta. O artigo escrito trata sobre o desenvolvimento de um *framework* para evoluir o algoritmo PSO utilizando Evolução Gramatical. Neste trabalho o objetivo é conseguir projetar um PSO de forma automática, procurando pela combinação de estruturas de parâmetros e operadores adequados (constantes de aceleração, equação de atualização da velocidade e topologia por exemplo).

A gramática que foi utilizada em [33], ver Figura 3.8, foi definida com a identificação de componentes considerados importantes para a geração de boas soluções no PSO [33]. Os componentes definidos foram tamanho da população, coeficientes de aceleração, probabilidade de ocorrer a mutação, fórmula de atualização da velocidade, dentre outras coisas. A combinação destes componentes caracteriza um projeto do algoritmo PSO, que é um possível indivíduo para a GE dentre os 2.041.200 possíveis, segundo os autores.

⟨*TAM*⟩ ::= 30 | 50 | 70

⟨*C1*⟩ ::= 1 | 1.25 | 1.75 | ... | 3

⟨*C2*⟩ ::= 1 | 1.25 | 1.75 | ... | 3

⟨*INERTIA*⟩ ::= 0.1 | 0.2 | ... | 1.0

⟨*PROB-MUTATION*⟩ ::= 0.1 | 0.2 | ... | 1.0

⟨*GBEST*⟩ ::= Global | Local | Focal | Von Neumann | Hierarchical | Four Clusters | Clan

⟨*UPDATE-VELOCITY*⟩ ::= (⟨*INERTIA*⟩ * v) + ⟨*C1*⟩ * r1 * (⟨*GBEST*⟩ - p) + ⟨*C2*⟩ * r2 * (pBest - p)
| X * (v + ⟨*C1*⟩ * r1 * (⟨*GBEST*⟩ - p) + ⟨*C2*⟩ * r2 * (pBest - p))

⟨*MUTATION*⟩ ::= Gaussian(⟨*PROB-MUTATION*⟩)
| Cauchy(⟨*PROB-MUTATION*⟩)
| Michalewicz(⟨*PROB-MUTATION*⟩)
| Levy(⟨*PROB-MUTATION*⟩)
| Random(⟨*PROB-MUTATION*⟩)
| λ

Figura 3.8: Gramática utilizada no trabalho de Miranda e Prudêncio [33].

A Figura 3.9 apresenta de forma geral o funcionamento do *framework* GEFPSO proposto por Miranda e Prudêncio. O primeiro passo é definir a gramática BNF. Depois disso, é gerada de forma aleatória uma população de indivíduos para o mecanismo de busca (GA). Depois, é realizado o processo de mapeamento e tradução dos indivíduos (vetores de inteiros) em programas (projeto do PSO) utilizando a gramática. A aptidão dos programas então é calculada com a execução dos programas correspondentes para um problema definido. Depois disso, é criado um laço que é executado enquanto um critério de parada não é atingido. Dentro do laço, é feita a seleção dos melhores indivíduos para serem utilizados na aplicação dos operadores de cruzamento e mutação para gerar indivíduos filhos. Nestes novos indivíduos é aplicado o processo de mapeamento para gerar os programas correspondentes e avaliar a aptidão dos mesmos. Quando o critério de parada é atingido, o algoritmo termina retornando o melhor projeto do PSO encontrado.

Para a execução dos projetos de PSO gerados pelo GEFPSO, foram utilizadas 16 funções de otimização, contínuas, sem constantes e com 30 dimensões. Estas funções estão associadas a quatro categorias: *Bowl-Shaped*, *Plate-Shaped*, *Valley-Shaped* e *Many Local Minima*, características que permitem avaliar a proposta do *framework* em diferentes níveis de dificuldade [33].

Este trabalho mostrou que algoritmos gerados de forma automática tem a capacidade de obter resultados comparáveis a algoritmos reconhecidos pela literatura. Uma característica que foi observada com relação as diferentes categorias das funções de *benchmark* utilizadas, é que um algoritmo gerado, desenvolvido e otimizado para ter um bom desempenho em um determinado problema, tem grande possibilidade de obter bons resultados em outros problemas da mesma categoria.

Levando em consideração o impacto que esse trabalho teve, pode-se perceber que ainda há outras oportunidades e conceitos para serem trabalhados e testados na ideia de geração

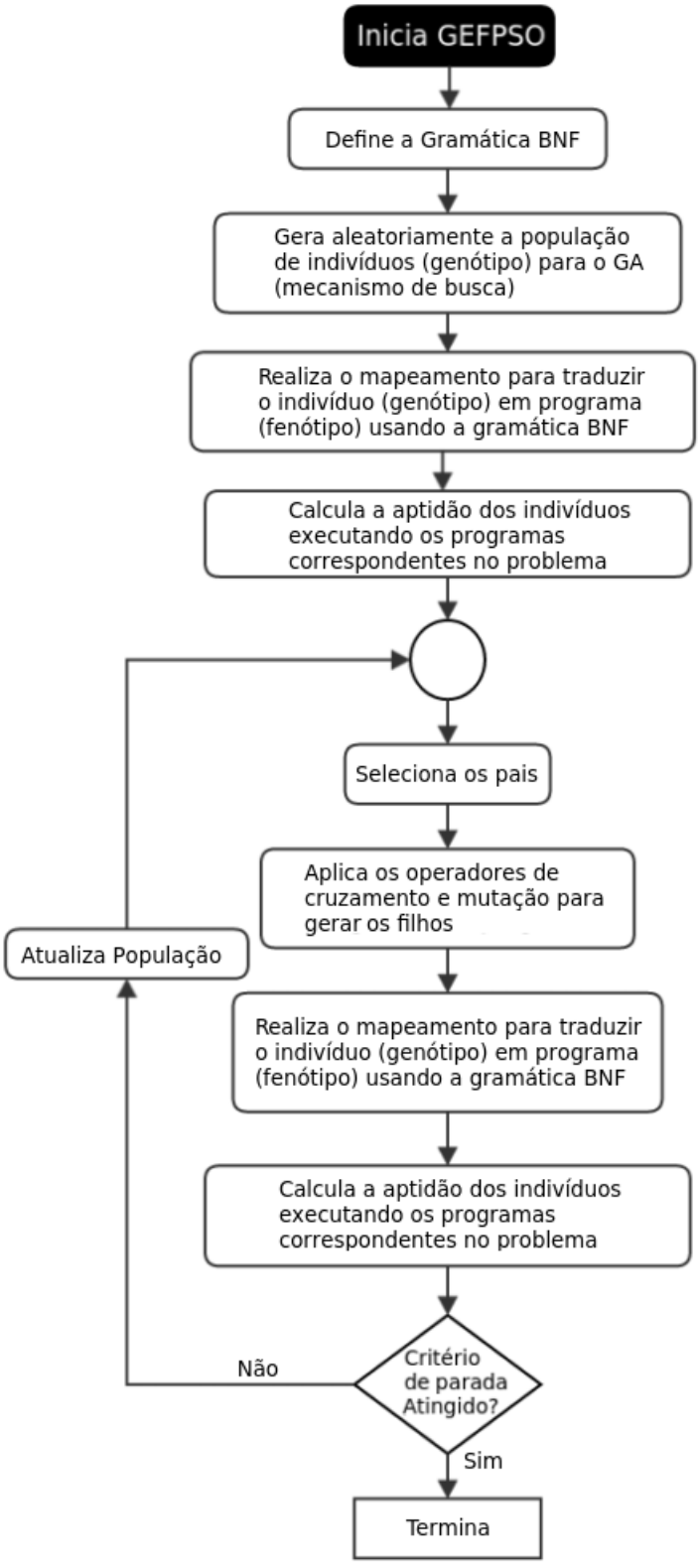

Figura 3.9: Esquema básico do *framework* GEFPSO. Adaptado de [33].

automática de algoritmos, sendo um exemplo tradado nessa proposta, explorar a possibilidade de aplicar os mesmos conceitos para algoritmos de otimização multiobjetivo, com diferentes componentes e estruturas.

Capítulo 4

IRACE

O **IRACE** [30] é um *framework* para configuração automática de algoritmos que considera diversos componentes algorítmicos como parâmetros categóricos a serem escolhidos. Nesse sentido, há um chamado espaço de configurações no qual o IRACE busca combinar diferentes componentes para gerar algoritmos únicos. Implementado no ambiente para computação estatística **R** [48], foi feito como uma extensão do chamado Iterated F-race (I/F-Race) proposto por Balaprakash *et al* em [2] e desenvolvido por Biratari *et al* em [4].

4.0.1 Configuração de Algoritmos

Muitos algoritmos para problemas de otimização são configuráveis, isto é, possuem um grande número de parâmetros que devem ser escolhidos pelo usuário. Como exemplo, nos algoritmos evolutivos deve-se especificar uma taxa de aplicação dos operadores de cruzamento e mutação [30]. O motivo pelo qual existem parâmetros configuráveis é por que não existe apenas uma combinação ótima, dependendo do problema que está sendo considerado. Dessa forma, a configuração de algoritmos pode ser tratada também como um problema de otimização onde se deve achar o conjunto ótimo de parâmetros que produza o melhor resultado possível quando aplicado ao algoritmo, como formulado por Birattari e Kacprzyk em [3].

Existem dois tipos principais de parâmetros: categórico e numérico. Parâmetros categóricos representam valores discretos sem qualquer ordem implícita ou medida de distância entre eles. Diferentes métodos de cruzamento são um exemplo em algoritmos evolutivos. Parâmetros numéricos possuem ordem implícita em seus valores. Valores de tamanho de população e taxa de mutação são exemplos. Há um terceiro tipo de parâmetro que se assemelha ao categórico, porém, possui um certo grau de ordem entre seus valores. Um exemplo seria um parâmetro com os três possíveis valores $\{baixo, médio, alto\}$. Estes parâmetros são chamados ordinais, e o IRACE os entende como parâmetros numéricos [30].

Parâmetros podem ser subordinados a outros parâmetros, isso quer dizer que eles somente são relevantes em determinados momentos dependendo do valor de outros parâmetros. Por exemplo, o operador de seleção pode receber os valores *roleta* ou *torneio*, porém, caso o valor utilizado seja o *torneio*, é necessário especificar um parâmetro extra, o "tamanho do torneio". Neste caso, o parâmetro *tamanho_torneio* é subordinado ao fato do parâmetro de seleção escolher o valor *torneio*. Parâmetros subordinados são o mesmo que constantes em relação ao valor de outros parâmetros. Por exemplo, dado dois parâmetros a e b, a constante pode ser $a < b$. Constantes limitam o alcance dos valores que um certo parâmetro pode tomar independente de outros, enquanto que parâmetros subordinados podem estar ativos ou até ter um valor de acordo com um alcance pré-definido [30].

4.0.2 Corrida Iterativa

A proposta do IRACE foi criada como uma implementação da corrida iterativa (*iterated racing*), I/F-Race [2] que utiliza o teste estatístico de Friedman (*Friedman's non-parametric two-way analysis of variance by ranks*).

A corrida iterativa é um método para a configuração automática que consiste em três passos: 1) experimentar novas configurações de acordo com uma distribuição em particular; 2) selecionar a melhor configuração das amostras recém testadas por meio de uma corrida; e 3) atualizar a distribuição das amostras de configurações com objetivo de guiar a experimentação em direção a melhor configuração. Estes três passos são repetidos até que um critério de parada seja satisfeito [2].

Na corrida iterativa, cada configuração possui um valor a partir de uma distribuição, sendo essa distribuição normal para parâmetros numéricos ou discreta para parâmetros categóricos. A atualização das distribuições é feita com a modificação das amostras das distribuições, a média e desvio padrão para a distribuição normal, ou probabilidades discretas dos valores da distribuição discreta. Esse processo é feito para aumentar a possibilidade das amostras criadas a partir das melhores configurações sejam mais escolhidas em iterações futuras [30].

A corrida se inicia com uma quantidade finita de configurações candidatas. A cada passo, as configurações são avaliadas em uma única instância. Após cada passo, através do teste estatístico são identificadas as piores configurações para descartá-las, continuando a corrida com as remanescentes até que um número máximo de instâncias seja utilizado ou esgotado um valor orçado (*budget*) de custo computacional. Esse valor de custo computacional pode ser no geral o tempo utilizado ou número de experimentos [30].

4.0.3 O pacote IRACE

O pacote IRACE compreende alguns componentes que juntos compõem o processo de otimização. A Figura 4.1 descreve como as diferentes partes do IRACE interagem entre si. O IRACE é composto de três entradas principais:

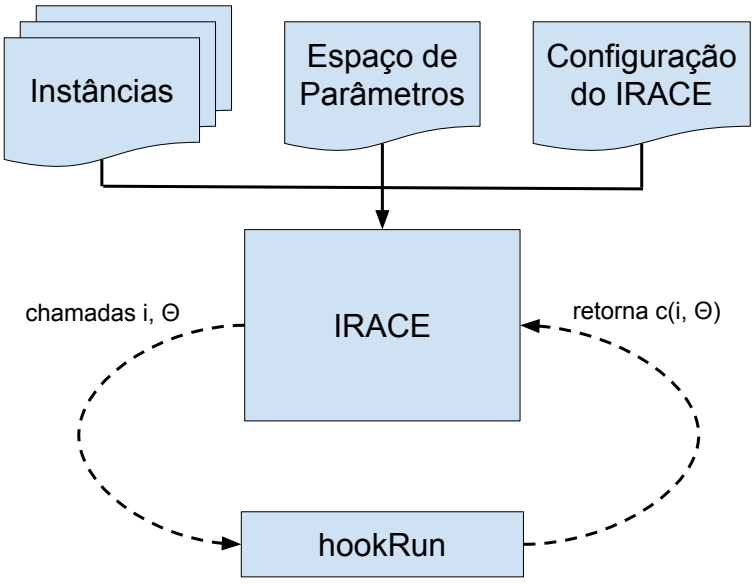

Figura 4.1: Esquema de fluxo de informações do IRACE. Adaptado de [30]

1. Descrição do espaço de parâmetros X, isto é, os possíveis parâmetros que podem ser configurados, seus tipos, valores e constantes.

2. O conjunto de instâncias de otimização $\{I_1, I_2, ...\}$, que é um conjunto finito usado para representar o problema tratado.

3. A configuração do IRACE, definida por um número de opções, tal como arquivo de definição dos parâmetros, *budget* de execução, número de candidatos, entre outros.

Para a descrição do espaço de parâmetros, o IRACE utiliza um arquivo de texto chamado por padrão de *parameters.txt*. Nele são definidos todos os possíveis parâmetros que podem ser utilizados para configurar o algoritmo em teste. Cada parâmetro é definido em uma linha deste arquivo, sendo composto de basicamente 5 colunas como mostra a Figura 4.2: identificador para o parâmetro, e como ele é referenciado pelo IRACE para o usuário, uma chave que identifica o parâmetro que será processada pelo módulo de execução (*hookRun*), o tipo de parâmetro sendo *c* para categóricos, *r* para números reais e *i* para inteiros. A quarta coluna descreve os possíveis valores para o parâmetro, e por fim, a quinta pode incluir condições para ativar ou não o parâmetro usando sintaxe do ambiente R.

```
# name          switch              type    values                  [conditions (using R syntax)]

algorithm       "--"                c       (as,mmas,eas,ras,acs)
localsearch     "--localsearch "    c       (0, 1, 2, 3)
alpha           "--alpha "          r       (0.00, 5.00)
beta            "--beta "           r       (0.00, 10.00)
rho             "--rho  "           r       (0.01, 1.00)
ants            "--ants "           i       (5, 100)
nnls            "--nnls "           i       (5, 50)                 | localsearch %in% c(1, 2, 3)
q0              "--q0 "             r       (0.0, 1.0)              | algorithm %in% c("acs")
dlb             "--dlb "            c       (0, 1)                  | localsearch %in% c(1,2,3)
rasrank         "--rasranks "       i       (1, 100)                | algorithm %in% c("ras")
elitistants     "--elitistants "    i       (1, 750)                | algorithm %in% c("eas")
```

Figura 4.2: Exemplo de parametrização do IRACE para o problema do Caixeiro Viajante.

As instâncias de problemas são arquivos de texto que ficam dentro de uma pasta chamada *Instances*. O conteúdo que cada arquivo de instância tem é dependente do problema tratado, pois, o IRACE somente passa o caminho do arquivo para o módulo *hookRun* como parâmetro para o algoritmo. Um exemplo seria as coordenadas das cidades consideradas em uma instância para o problema do caixeiro viajante.

As configurações do IRACE são armazenadas no arquivo *tune-conf.txt*. Nele, o usuário pode definir diversas opções oferecidas, como por exemplo o nome do arquivo de parametrização (*parameters.txt*), a pasta que são criados os arquivos temporários (chamada “arena”), a pasta de instâncias, o número máximo de experimentos, precisão dos valores, dentre outros.

O *hookRun* é um módulo usado pelo IRACE responsável por receber e passar as configurações amostradas durante o processo de busca para o algoritmo que está sendo otimizado. As configurações são repassadas ao algoritmo em forma de uma lista de argumentos de acordo com o especificado no arquivo *parameters.txt*. Os resultados obtidos pelo algoritmo testado na configuração passada como parâmetros é devolvida ao IRACE como aptidão.

Capítulo 5

Trabalhos Relacionados

Para o desenvolvimento do trabalho proposto foi necessário pesquisar sobre os assuntos relacionados e elicitar os principais trabalhos que contribuíssem para o tema, bem como qualquer informação relevante que auxilie no processo. Os trabalhos mais relevantes estão descritos a seguir.

O principal trabalho sobre MOPSO é de Coello e Lechuga [12] que propõe o algoritmo. Este trabalho introduz o conceito da dominância de Pareto ao PSO, para auxiliar as partículas a determinar a direção de vôo e a manter soluções não-dominadas em um arquivo externo. A principal contribuição deste trabalho está na apresentação da estrutura do algoritmo, como ele funciona, seus componentes, parâmetros e estruturas que são fatores importantes para ajudar no planejamento e desenvolvimento da gramática. Mais detalhes sobre o algoritmo em si foram dadas na Seção 2.4.

No Livro *A Field Guide to Genetic Programming* de Poli *et al.*[43] está explicado grande parte do conhecimento necessário para entender sobre programação genética. O livro aborda desde os princípios básicos, representação das soluções, operadores, avaliação, parâmetros, até técnicas avançadas para PG, inicialização, *seeding*, estruturas modulares, aplicação da gramática, e também sobre a aplicação dos conceitos em aplicações práticas.

O trabalho de Ryan, Collins e O'Neill [45] apresenta conceitos sobre a utilização da Evolução Gramatical para evoluir programas a partir de uma linguagem arbitrária. São apresentadas as vantagens e o crescente uso da programação genética para a evolução de programas de computador. Explicando a proposta da utilização de gramáticas livres de contexto em conjunto com a PG para evoluir código. E como o sistema usa árvores sintáticas que são geradas a partir da gramática definida.

Em um trabalho proposto por Lourenço, Pereira e Costa [31], é abordada a criação de um *framework* para evolução de algoritmos evolutivos utilizando evolução gramatical. Neste trabalho é definida uma gramática com base nos componentes do algoritmo. Como o objetivo era evoluir um algoritmo genético, os componentes consistiam em selecionar tipo da população, método de seleção, cruzamento, mutação, elitismo e taxas de operadores. A partir dos experimentos conduzidos, foi constatado que alguns projetos não-usuais obtiveram resultados competitivos com abordagens consideradas padrão.

Em um artigo de Castro Jr e Pozo [9] os autores propõem a utilização de uma Hiper-Heurística para automaticamente selecionar os operadores de Arquivamento e Seleção de Líder para o MOPSO. A motivação foi que, dependendo do problema, diferentes operadores de arquivamento e seleção de líder tem melhor desempenho, variando também em que estágio da busca ele é aplicado. Com base nos experimentos realizados pelos autores, quando comparado

com outros algoritmos do estado-da-arte como o MOEA/D-DRA, o algoritmo proposto se mostra robusto e com bons resultados em diferentes situações.

No trabalho de Contreras-Bolton e Parada [13] é proposta a utilização de programação genética para a evolução de algoritmos de *clustering* e também para *Traveling Salesman Problem*. No trabalho é descrito como foram definidas as funções que descrevem rotinas dos algoritmos que se quer evoluir, definidas como nós para as árvores sintáticas da PG. Para os experimentos conduzidos no trabalho, os resultados mostram que o desempenho dos algoritmos evoluídos, apesar de conseguirem bons resultados em alguns casos, para problemas com características similares, algoritmos mais especializados são necessários.

Miranda e Prudêncio [33] propõe um *framework* chamado GEFPSO (de *Grammatical Evolution for PSO*) para evoluir o PSO utilizando evolução gramatical, sendo um dos trabalhos usados como base para esta proposta. Neste trabalho é definida uma gramática para os componentes do PSO, como população, mutação e atualização da velocidade, dentre outros. A GE fica encarregada de evoluir projetos criados para o PSO com o objetivo de encontrar configurações de forma mais barata e rápida do que se fossem configuradas manualmente. Os resultados obtidos com experimentos mostram que alguns dos algoritmos obtidos possuem bom desempenho ao longo da aplicação em diferentes problemas, onde o algoritmo conseguiu obter resultados competitivos quando comparados com outros algoritmos bem sucedidos da área.

Dentre todos os trabalhos citados, pode-se perceber que por mais promissores que os resultados obtidos por um algoritmo heurístico possam ser, é difícil afirmar que a configuração utilizada seja a melhor possível, sendo necessária nesses casos a aplicação de alguma técnica para tentar tirar o melhor proveito possível, como acontece em certos casos, onde são utilizadas hiper-heurísticas como no trabalho de Castro e Pozo em [8], e em outros aplicando programação genética como Contreras-Bolton e Parada em [13] e Miranda e Prudêncio [33]. Para esses trabalhos, o principal objetivo é mostrar que evoluir o código como um todo pode ser mais vantajoso e também em alguns casos consegue gerar algoritmos que obtenham melhores resultados que algoritmos desenhados de forma manual. O grande problema por trás das abordagens com PG se dá pela dificuldade na escolha de componentes que constituem um programa. Além do cuidado necessário para avaliar se o programa resultante não se tornou super especializado para um determinado problema.

Capítulo 6

Projeto automático de algoritmos MOPSO

Baseado no que foi apresentado nas seções anteriores sobre configuração automática, algumas oportunidades de pesquisa surgiram sobre como o PSO pode ser modificado para buscar melhores resultados através da utilização de uma técnica que o auxilie a configuração de seus componentes. Porém, diferente do trabalho de Castro e Pozo [9], que utiliza uma abordagem de Hiper-Heurística para configurar dinamicamente os componentes do MOPSO, levando em consideração que recentes estudos na área mostraram que ao invés de configurar componentes, seria mais interessante projetar o algoritmo, selecionando componentes e estruturas de código, como é apresentado no trabalho de Contreras-Bolton e Parada [13] no uso da PG para o problema de *Clustering* e Miranda e Prudêncio [33] na aplicação de GE para evoluir o PSO.

Considerando o estado da arte, este trabalho propôs a aplicação dos conceitos abordados no trabalho de Miranda e Prudêncio em [33], para utilizar Evolução Gramatical no algoritmo MOPSO, expandindo os estudos sobre projeto automático de algoritmos. A principal diferença deste trabalho para os anteriores é a aplicação do método em um algoritmo multiobjetivo (no caso o MOPSO) além da criação de uma nova gramática com inclusão dos componentes que o PSO não possui, como operadores de arquivamento e seleção de líder.

A ideia por trás do método proposto é simples. O *framework* proposto, chamado GEMOPSO, oferece módulos para realizar a criação e otimização de algoritmos de forma automática. Os principais módulos são as metodologias de treinamento e teste, nos quais são criados, treinados e testados os projetos de MOPSO com base nos problemas considerados e componentes da gramática. Para o treino é possível selecionar dois algoritmos diferentes: GE ou IRACE. A GE cria e melhora os projetos de MOPSO em um processo evolutivo com aplicação de operadores de cruzamento mutação; e o IRACE seleciona os melhores projetos por meio de uma corrida entre diferentes algoritmos, selecionando os melhores estatisticamente diferentes dos demais.

Este capítulo descreve a aplicação dos conceitos e ideias apresentadas anteriormente para implementação e teste da viabilidade e desempenho do GEMOPSO. Ao longo do capítulo são detalhados os módulos e funcionamento do *framework*, bem como a representação dos algoritmos, gramática utilizada, diferentes algoritmos de projeto e avaliação das soluções.

6.1 Algoritmo MOPSO

O Algoritmo base do MOPSO é representado pelo Algoritmo 6. Cada projeto individual é construído a partir da adição dos parâmetros e componentes ao algoritmo base. Dentre todos os possíveis componentes e parâmetros disponíveis para o MOPSO, foram escolhidos com base em outros trabalhos relacionados e experimentos preliminares os seguintes: tamanho da população

(Linha 1), tipo de repositório (Linha 2), operador de seleção de líder (Linha 6), operador de velocidade (Linha 9), operador de mutação (Linha 13) e método de atualização do repositório (Linhas 18 e 22). O componente evolução (EVOL, Linha 7), refere-se a modificação realizada no algoritmo para permitir a criação de projetos não usuais do MOPSO como por exemplo o uso de mais operadores de velocidade e/ou mutação.

Algoritmo 6: Esqueleto básico do MOPSO para o GEMOPSO

```
1  enxame = cria populacao (TAM);
2  inicia o repositório (REP);
3  avalia repositório;
4  enquanto critério de parada não satisfeito faça
5      para cada partícula ∈ enxame faça
6          seleção de líder(LIDER);
7          para cada componente ∈ EVOL faça
8              se componente = VEL então
9                  atualiza velocidade (VEL);
10                 atualiza posição;
11             fim
12             senão se componente = MUTACAO então
13                 aplica mutação (MUTACAO);
14             fim
15         fim
16         avalia partícula;
17         atualiza pBest;
18         se REP_ATT = inLoop então
19             atualiza repositório;
20         fim
21     fim
22     se REP_ATT = outLoop então
23         atualiza repositório;
24     fim
25 fim
26 retorna repositório;
```

A avaliação de cada projeto MOPSO é representada pelo valor médio para um dado número de execuções. Diferentes componentes produzem diferentes comportamentos por parte do algoritmo. Essa diferença permite escolhas que focam em convergência, manter soluções bem espalhadas ou outras características dependendo das preferências.

6.2 Componentes e Parâmetros

A Evolução Gramatical permite a partir de uma gramática a criação de algoritmos baseados na escolha de componentes e parâmetros, por isso, é importante ter bem definido e estruturado como o algoritmo será criado e quais serão os componentes e parâmetros que farão parte da gramática utilizada.

Para facilitar a estruturação do MOPSO para usá-lo com a GE, o algoritmo deve ser construído a partir da união de diferentes componentes, como se fossem blocos de montar, que

podem ser encaixados de diferentes formas. Dessa forma, fica mais simples modificar o algoritmo sem ter que reestruturar todo o código, mantendo o foco nos componentes que causam maior impacto. É nesse sentido que o trabalho de Miranda e Prudêncio [33] cria uma gramática para o PSO que era composta por alguns parâmetros como tamanho da população, coeficientes de velocidade e método de mutação, dentre outros.

Os componentes selecionados para o GEMOPSO são descritos a seguir:

- **Tamanho da população**: os possíveis valores para este parâmetro foram escolhidos baseados em trabalhos anteriores. O número de soluções na população afeta a convergência e qualidade das soluções retornadas pelo MOPSO.
- **Repositório**: os métodos de repositório foram definidos baseados nos métodos mais utilizados para o MOPSO. Dentre eles, tem-se *Crowding Distance* [18], *Ideal* [5] e *Multilevel Grid* [29]. O repositório armazena as melhores soluções não-dominadas para o conjunto final de soluções. Ele também toma como parâmetro adicional a forma que é feita a atualização do operador, seja após a avaliação de cada partícula, ou após todas passarem pelo processo evolutivo.
- **Seleção de Líder**: estão sendo considerados três métodos conhecidos para auxiliar na escolha do $gBest$ durante o processo evolutivo. São eles *Crowding Distance* [18], *Sigma* [35] e *NWSum* [39].
- **Atualização de Velocidade**: para este operador, foram considerados métodos usados no PSO tradicional *Standard*, algumas variações como *Inertia* e *Constriction*, mas também o operador de velocidade utilizado no SMPSO [36], chamado *Speed Constraint*. Há um parâmetro extra chamado Inércia, utilizado somente quando o operador de velocidade é *Inertia* ou *Speed Constraint*.
- **C1** e **C2**: Os coeficientes C_1 e C_2 são utilizados durante a atualização da velocidade das partículas como forma de balancear o peso dos melhores local e global.
- **Mutação**: foram considerados para o operador de mutação os melhores métodos da literatura, o método *Polynomial* e *Uniform*. Como parâmetro adicional, a probabilidade da mutação ocorrer.

A Gramática 6.1 foi criada utilizando os componentes listados. A partir das regras de produção é possível gerar combinações entre os componentes definidos para dar forma a um projeto do MOPSO.

Utilizando a gramática de [33] como base, definimos que um MOPSO é basicamente composto pelos seguintes componentes: tamanho da população, operador de repositório, operador de seleção de líder, operador de velocidade e operador de mutação. Para os operadores de arquivamento, há a regra que $< REP\_ATT >$ utilizada para definir como será feita a atualização do repositório. Da mesma forma para as regras $< VEL >$ e $< MUTACAO >$, possuem parâmetros extras. As variáveis C_1 e C_2 são definidas como intervalos. Por exemplo, 1.25-2.25 para C_1 indica que durante a aplicação do operador C_1 terá seu valor escolhido de forma aleatória entre o intervalo de 1.25 e 2.25.

A regra $< EVOL >$ representa a principal mudança entre as duas propostas para esta gramática. Utilizando a regra $< EVOL >$ normal, podemos considerar que os MOPSOs capazes de serem gerados seguem a estrutura tradicional do algoritmo, pois é escolhido um componente de cada tipo para aplicar ao algoritmo base do MOPSO (Algoritmo 6). Utilizando a regra

⟨*MOPSO*⟩ ::= ⟨*TAM*⟩ ⟨*REP*⟩ ⟨*LIDER*⟩ ⟨*EVOL*⟩

⟨*TAM*⟩ ::= 30 | 50 | 70

⟨*REP*⟩ ::= Crowding Distance ⟨*REP_ATT*⟩ | Ideal ⟨*REP_ATT*⟩ | Multilevel Grid Archiving ⟨*REP_ATT*⟩

⟨*REP_ATT*⟩ ::= In Loop | Out Loop

⟨*LIDER*⟩ ::= Crowding Distance | Sigma | NWSUM

⟨*EVOL*⟩ ::= ⟨*VEL*⟩ ⟨*MUTACAO*⟩

⟨*EVOL-R*⟩ ::= ⟨*VEL*⟩ ⟨*EVOL-R*⟩ | ⟨*MUTACAO*⟩ ⟨*EVOL-R*⟩ | ⟨*VEL*⟩ | ⟨*MUTACAO*⟩

⟨*VEL*⟩ ::= Standard ⟨*C1*⟩⟨*C2*⟩ | Constriction ⟨*C1*⟩⟨*C2*⟩ | Inertia ⟨*C1*⟩⟨*C2*⟩⟨*INERCIA*⟩

⟨*C1*⟩ ::= 1-2 | 1.25-2.25 | 1.5-2.5 | 1.75-2.75 | 2-3

⟨*C2*⟩ ::= 1-2 | 1.25-2.25 | 1.5-2.5 | 1.75-2.75 | 2-3

⟨*INERCIA*⟩ ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1

⟨*MUTACAO*⟩ ::= Polynomial ⟨*PROB*⟩ | Uniform ⟨*PROB*⟩

⟨*PROB*⟩ ::= 0.01 | 0.025 | 0.05 | 0.075 | 0.1 | 0.125 | 0.15 | 0.175 | 0.2

Figura 6.1: Gramática proposta para o GEMOPSO.

$< EVOL - R >$ substituindo a regra original, é possível expandir a quantidade de possibilidades de algoritmos gerados, pois a regra $< EVOL - R >$ pode ser aplicada recursivamente, sem um limite essa regra poderia ser aplicada infinitamente, gerando MOPSOs com diferentes operadores de velocidade e mutação.

6.3 Instâncias de Treino

Para executar os projetos MOPSO é necessário um conjunto de instâncias de treinamento. Dependendo do algoritmo de projeto usado (GE ou IRACE), as instâncias são aplicadas de forma diferente. Para a GE, cada MOPSO deve ser executado várias vezes para cada instância de treinamento, produzindo um valor de média para cada instância e no fim a média para todas as instâncias. Para o IRACE, a cada passo do processo de busca, uma instância é selecionada de forma aleatória para ser aplicada a cada MOPSO, como resultado, somente os melhores projetos de MOPSO sobrevivem através das gerações.

6.4 Projeto de Algoritmos

Ambas as estratégias GE e IRACE possuem o mesmo propósito, que é a geração de MOPSOs. A GE se baseia em um processo evolutivo, já o IRACE utiliza um método de busca baseado em uma “corrida”. Entretanto, cada estratégia possui diferentes requisitos e elementos para o projeto dos algoritmos.

A GE usa um motor de busca para evoluir uma população de projetos MOPSO através de gerações e aplicação dos operadores de seleção, cruzamento e mutação para manipular as

soluções. Normalmente é utilizado um GA (Algoritmo 7) como algoritmo de evolução das soluções, sendo necessária somente a adição da etapa de mapeamento que ocorre antes de avaliação de soluções, e de dois novos operadores, duplicação e *pruning*.

Uma gramática específica é usada para definir as regras de produção para o processo de mapeamento. Também inclui os diferentes componentes e parâmetros usados pelo MOPSO tradicional e também para a versão recursiva. No fim, o algoritmo retorna como saída o melhor projeto encontrado durante a evolução.

Algoritmo 7: Evolução Gramatical

```
1  criar população de soluções de tamanho variável;
2  mapeamento;
3  avaliação da população;
4  enquanto critério de parada não satisfeito faça
5      seleciona pais;
6      filhos ← aplica cruzamento;
7      aplica mutação para filhos;
8      aplica duplicação para filhos;
9      aplica prune para filhos;
10     mapeamento;
11     avaliação da população;
12     substituição;
13 fim
14 retorna melhor solução encontrada;
```

6.4.1 Avaliação de soluções

Avaliar o desempenho de um algoritmo que evolui outros algoritmos ou programas pode ser uma tarefa custosa, pois cada solução deve ser executada com seus próprios parâmetros, e cada caso pode variar em tempo de execução de acordo com os componentes utilizados.

Buscando facilitar a avaliação dos indivíduos, a execução dos algoritmos gerados é feita para problemas de otimização conhecidos, tornando mais justa a comparação dos desempenhos.

O *Hypervolume*, representado na Figura 6.2, é um indicador de qualidade proposto nos estudos de [51], denotado como o "tamanho da área coberta no espaço de objetivos". Este indicador consegue medir a partir de um ponto escolhido como sendo o pior possível, o quanto um conjunto de soluções preenche o espaço de objetivos. Quanto maiores os valores obtidos pelo *hypervolume*, maior é a cobertura do espaço, e consequentemente melhor é a qualidade do conjunto avaliado.

A escolha do indicador de qualidade *Hypervolume* se deu principalmente pela sua característica, em que as menores mudanças no conjunto de soluções produzem visíveis alterações no valor de *hypervolume*, o que facilita na análise dos resultados.

6.4.2 Operadores

Além dos operadores definidos para serem usados no MOPSO, também é necessário definir os operadores que serão utilizados no mecanismo de busca da GE, nesse caso um GA. São esses operadores que modificam e introduzem pequenas mudanças nas soluções para produzir

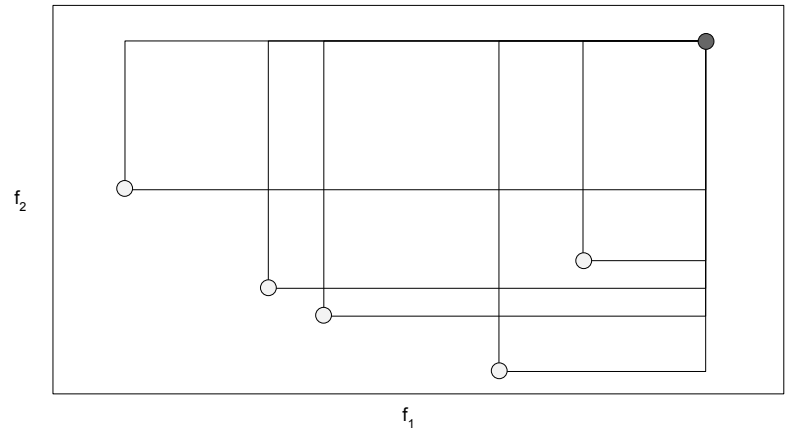

Figura 6.2: Exemplo do indicador de *Hypervolume*.

diferentes projetos do MOPSO, gerando maior diversidade. Os operadores que estão sendo propostos foram selecionados com base em [6, 7, 31, 33, 38], dentre outros.

Como as soluções manipuladas pela GE são vetores de inteiros, existem diversas técnicas diferentes para se trabalhar com esse tipo de indivíduo. Por esse motivo, foram selecionadas algumas técnicas com base nos melhores operadores disponíveis na literatura. Os operadores selecionados foram:

- **Cruzamento SinglePoint:** A partir de um ponto que é escolhido de forma aleatória, todos os dados além do ponto são trocados entre os indivíduos pais selecionados. O resultado são dois indivíduos filhos, assim como mostra a Figura 6.3.

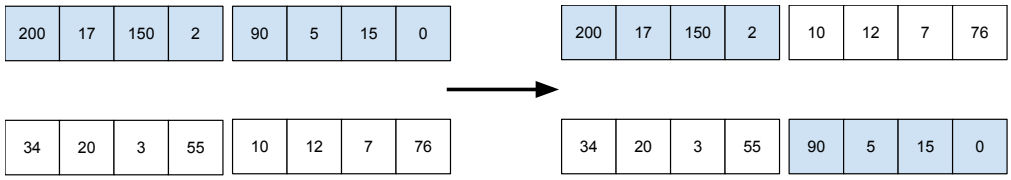

Figura 6.3: Exemplo de aplicação do operador OnePoint.

- **Cruzamento TwoPoints:** Semelhante ao anterior, a partir de dois pontos escolhidos aleatoriamente, todos os dados entre os dois pontos são trocado entre os indivíduos pais, dando origem a dois indivíduos filhos, assim como mostra a Figura 6.4.

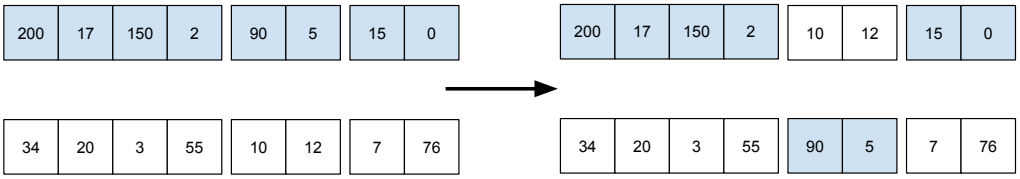

Figura 6.4: Exemplo de aplicação do operador TwoPoints.

- **Mutação BitFlip:** Este método de mutação consiste em atribuir a cada posição do indivíduo uma chance de seu valor ser alterado. Para cada posição onde a chance atribuída ultrapassar um dado parâmetro, o valor daquela posição é invertido se o indivíduo for

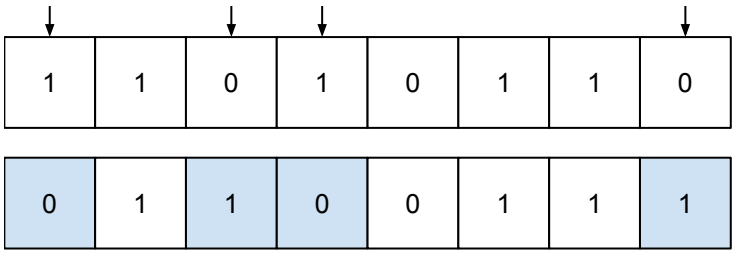

Figura 6.5: Exemplo de aplicação do operador BitFlip. As posições marcadas foram selecionadas para serem invertidas.

binário, como mostra a Figura 6.5, ou substituído por um novo valor aleatório para indivíduos com valores inteiros, respeitando limites inferior e superior.

- **Binary Tournament:** O torneio binário é um método de seleção muito popular. Nele são selecionados dois indivíduos de forma aleatória, a partir disso, os dois são comparados e o melhor dentre eles é escolhido, assim como mostra a Figura 6.6.

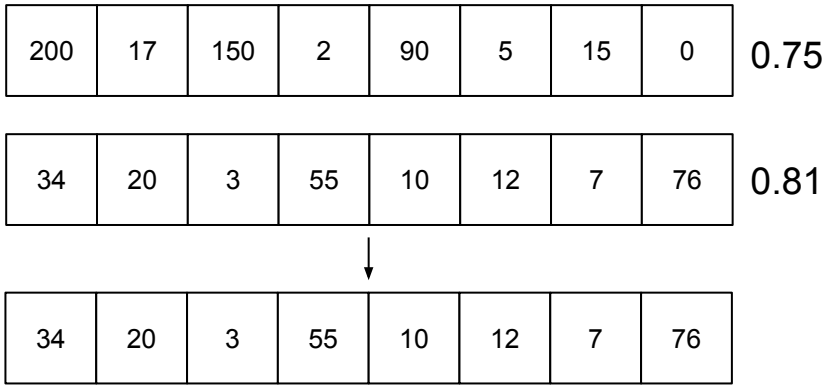

Figura 6.6: Exemplo de aplicação do operador torneio binário. Comparando a aptidão dos indivíduos, o segundo seria escolhido para um problema de maximização.

Na GE também existem dois outros operadores que são utilizados quando se usa indivíduos de tamanho variável, são eles os operadores de duplicação e *pruning*.

- **Duplicação:** O operador de duplicação envolve aleatoriamente selecionar uma quantidade de códons do início do vetor para colocá-los no fim [38].

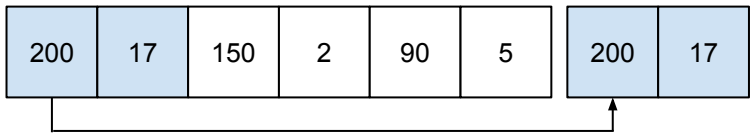

Figura 6.7: Exemplo de aplicação do operador de duplicação. Os dois primeiros genes são copiados para o fim do indivíduo.

- **Pruning:** Esse operador tem como objetivo remover genes que estão sobrando em um indivíduo. Se o indivíduo gerou algo válido com apenas 7 dos 10 genes, os 3 sobrando são descartados [45].

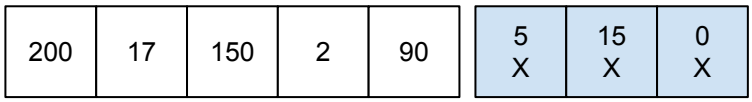

Figura 6.8: Exemplo de aplicação do operador de Pruning. O indivíduo só precisou de 5 dos 8 genes, ignorando os 3 últimos.

O treinamento é um dos principais módulos no GEMOPSO. Sendo responsável pela criação e melhora dos projetos MOPSO de acordo com a técnica e problemas empregados.

De acordo com a Figura 6.9, podemos analisar como se dá o processo de treinamento de forma simplificada. O GEMOPSO precisa que sejam definidos quais são os problemas nos quais os algoritmos serão treinados, e também a gramática que será empregada nas técnicas. Há uma diferença em como a gramática é configurada para a GE e para o IRACE, enquanto a GE lê um arquivo de texto da gramática em formato BNF, o IRACE precisa que as regras de produção sejam adicionadas conforme o padrão do IRACE no arquivo de configuração.

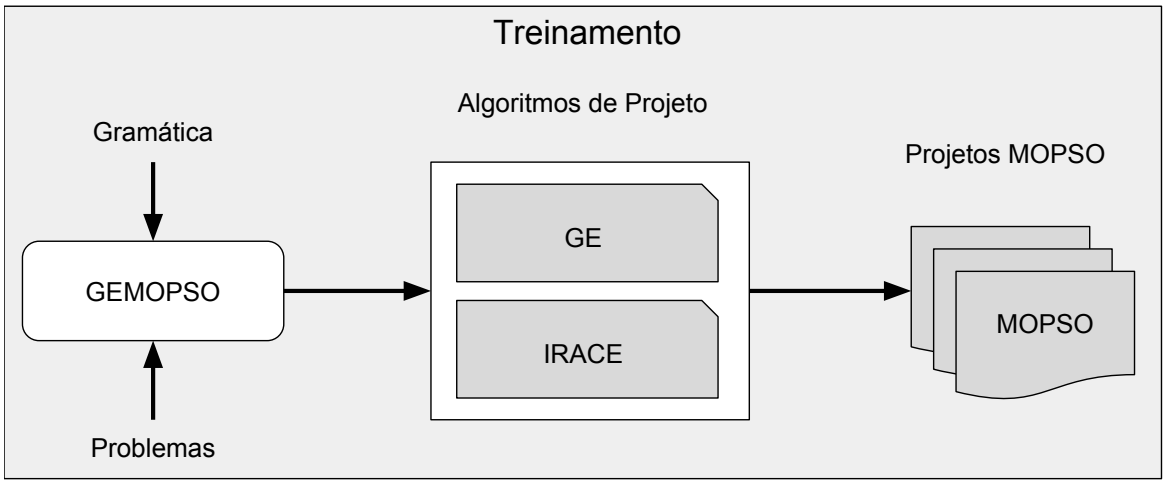

Figura 6.9: Exemplificação do processo de treinamento do GEMOPSO.

Capítulo 7

Experimentos

A experimentação é uma importante etapa utilizada para avaliar se um determinado método se encaixa com a proposta criada para ele. Este capítulo apresenta como foi organizada a fase de experimentação do GEMOPSO com objetivo de comparar seus resultados e desempenho com outros algoritmos.

Os experimentos são constituídos de basicamente duas fases: treinamento e teste, que são utilizados para criar e avaliar os MOPSOs de acordo com um conjunto de cenários baseados em um *benchmark*. Os experimentos foram divididos em três etapas principais. A primeira etapa compreende a comparação de duas versões da gramática proposta. A segunda, compara os dois métodos de projeto de algoritmos presentes no GEMOPSO. A etapa final consiste em comparar os melhores projetos de MOPSO gerados com o SMPSO, algoritmo conhecido na literatura. Os experimentos são aplicados para as versões normal e recursiva da gramática de forma individual.

7.1 Cenários

Um conjunto de *benchmark* é normalmente utilizado para testar um método, sob várias instâncias diferentes de problemas conhecidos, para avaliar seu desempenho. Para estes experimentos, foram consideradas as funções DTLZ. O *benchmark* DTLZ [19] foi originalmente proposto como forma de melhorar o já existente *benchmark* ZDT [50] permitindo escalabilidade no número de objetivos considerados nos problemas, como mostra um exemplo nas figuras 7.1 e 7.2. O conjunto é composto por sete funções chamadas DTLZ1-7, seu uso deu pela sua ampla utilização na literatura e suas características que buscam explicitar as dificuldades enfrentadas pelos algoritmos, como a convergência para à Fronteira de Pareto-Ótima e manutenção de uma variedade de soluções bem distribuídas.

Foram criados dois grupos de cenários utilizando o conjunto de *benchmark* DTLZ. Cada grupo é composto de 6 cenários, sendo 3 para a fase de treinamento e 3 para teste. No primeiro grupo, os Cenários de Treinamento combinam as funções DTLZ1-7 juntamente com o conjunto de variáveis $V\{30, 40, 50\}$ e o conjunto de objetivos $N\{2, 3, 5\}$. Os Cenários de Teste combinam as funções DTLZ1-7, o conjunto de variáveis $N_{test}\{20, 60\}$ e o mesmo conjunto de objetivos N. O segundo grupo é semelhante ao primeiro, porém, ao invés de considerar as funções DTLZ1-7, utiliza apenas as funções DTLZ1-6. A DTLZ7 foi desconsiderada após ser constatado durante os experimentos da primeira etapa que essa função parecia estar causando um impacto aparentemente negativo durante a fase de treinamento, tornando os algoritmos tendenciosos. As tabelas 7.1 e 7.2 representam os cenários criados, sendo nomeados com A os cenários que utilizam as funções DTLZ1-7 e B que utilizam as funções 1-6.

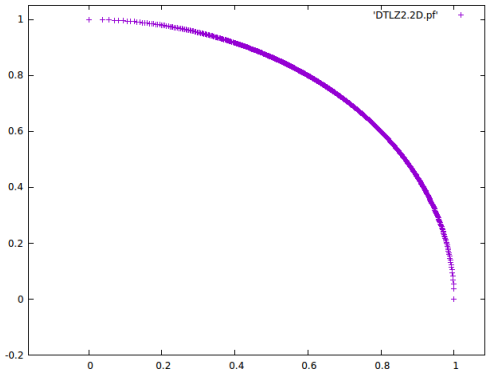

Figura 7.1: Fronteira do Problema DTLZ2 para dois objetivos.

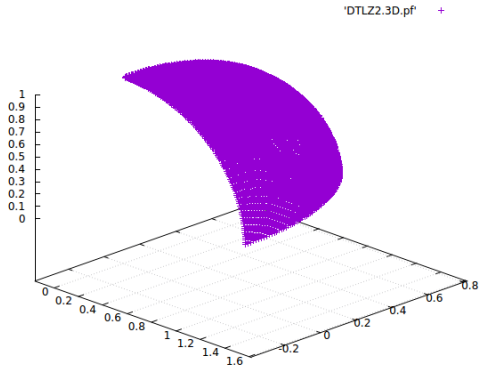

Figura 7.2: Fronteira do Problema DTLZ2 para três objetivos.

Tabela 7.1: Cenários propostos para experimentos de treino

Cenário	Problemas	Variáveis	Objetivos
1A	DTLZ1-7	30, 40, 50	2
2A	DTLZ1-7	30, 40, 50	3
3A	DTLZ1-7	30, 40, 50	5
1B	DTLZ1-6	30, 40, 50	2
2B	DTLZ1-6	30, 40, 50	3
3B	DTLZ1-6	30, 40, 50	5

Tabela 7.2: Cenários propostos para experimentos de teste

Cenário	Problemas	Variáveis	Objetivos
1A	DTLZ1-7	20, 60	2
2A	DTLZ1-7	20, 60	3
3A	DTLZ1-7	20, 60	5
1B	DTLZ1-6	20, 60	2
2B	DTLZ1-6	20, 60	3
3B	DTLZ1-6	20, 60	5

7.2 Treinamento

O objetivo da fase de treinamento é submeter os algoritmos de projeto a um conjunto de experimentos utilizando as instâncias de treinamento e então gerar os melhores MOPSOs. As soluções são desenvolvidas para possuir um bom desempenho nas instâncias de treino, e também demonstrar comportamento similar quando executados nas instâncias de teste.

Cada algoritmo de projeto trabalha de forma diferente. A GE como algoritmo evolutivo aprimora as soluções através da aplicação dos operadores de cruzamento e mutação ao longo de um certo número de gerações. O IRACE obtém as melhores configurações para o MOPSO através da "corrida", onde as piores soluções vão sendo eliminadas passo a passo.

7.3 Teste

Após o treinamento, os melhores MOPSOs encontrados são então submetidos aos cenários de teste, os quais são usados para verificar se os algoritmos gerados tem o que é

necessário para manter um bom desempenho e comportamento consistente quando executados sob novas instâncias de problemas.

7.4 Metodologia

Durante os experimentos o objetivo é que os MOPSOs gerados tenham bom desempenho quando executados em diferentes situações. Por esse motivo, primeiro os algoritmos são executados em cenários de treino que originam MOPSOs. Depois, os MOPSOs gerados são executados nos cenários de teste. Os resultados obtidos são avaliados utilizando-se o indicador de qualidade *hypervolume* e comparados de acordo com as etapas definidas anteriormente:

O primeiro conjunto de experimentos compreende a comparação entre duas versões da gramática proposta, as alterações se aplicam tanto para a versão normal quanto para a recursiva. As principais diferenças são explicitas nos trechos das gramáticas 7.3 e 7.4 (referenciadas como A e B respectivamente) abaixo.

⟨*C1*⟩ ::= 1 | 1.25 | 1.5 | 1.75 | 2 | 2.25 | 2.5 | 2.75 | 3

⟨*C2*⟩ ::= 1 | 1.25 | 1.5 | 1.75 | 2 | 2.25 | 2.5 | 2.75 | 3

⟨*PROB*⟩ ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0

Figura 7.3: Trecho da gramática A proposta com regras C1, C2 e PROB baseadas no trabalho de Miranda e Prudêncio [33].

O trecho de gramática da Figura 7.3 representa a versão da Gramática proposta que manteve as regras $C1$, $C2$ e $PROB$ baseados no trabalho de Miranda e Prudêncio [33] onde $C1$ e $C2$ são escolhidos diretamente, e $PROB$ possui um intervalo dentre 10% e 100%.

⟨*C1*⟩ ::= 1-2 | 1.25-2.25 | 1.5-2.5 | 1.75-2.75 | 2-3

⟨*C2*⟩ ::= 1-2 | 1.25-2.25 | 1.5-2.5 | 1.75-2.75 | 2-3

⟨*PROB*⟩ ::= 0.01 | 0.025 | 0.05 | 0.075 | 0.1 | 0.125 | 0.15 | 0.175 | 0.2

Figura 7.4: Trecho da gramática B proposta com regras C1, C2 e PROB modificadas para este trabalho.

Já no trecho de gramática da Figura 7.4, as regras $< C1 >$ e $< C2 >$ foram definidas de maneira que fossem utilizados limitantes inferior e superior ao invés de valores fixos. Isso permite que, durante a aplicação do operador de velocidade, os valores de $C1$ e $C2$ sejam escolhidos de forma aleatória entre os limitantes selecionados durante o processo de mapeamento. De forma semelhante, a regra $< PROB >$ teve seus valores alterados para uma escala com crescimento de 2.5% e diminuindo o intervalo para valores entre 1% e 20%.

A comparação das gramáticas foi realizada utilizando os cenários A, que incluem a DTLZ7. Através deles foi possível identificar que essa função pode estar causando certa influência no treinamento, e fazendo com que os MOPSOs gerados tenham bom desempenho somente nesta função quando comparados com o SMPSO.

O segundo conjunto de experimentos realiza a comparação entre as técnicas de projeto de algoritmos presentes no GEMOPSO. Os experimentos foram realizados utilizando os cenários B (sem a DTLZ7) e depois comparados entre si. O algoritmo que obteve o melhor desempenho foi escolhido para ser comparado com o SMPSO.

O terceiro conjunto de experimentos focou na comparação dos melhores MOPSOs obtidos com o treinamento, com o algoritmo SMPSO. O SMPSO possui bom desempenho nos problemas considerados e está sendo usado como referência.

Os algoritmos GE e IRACE são executados 30 rodadas independentes, cada MOPSO é executado 10 vezes usando um máximo de 10000 avaliações. A aptidão dos algoritmos é calculada através do indicador de qualidade *hypervolume*, como a média de *hypervolume* para todas as 10 execuções. Tanto o GE quanto o IRACE utilizam 100 candidatos como tamanho da população. O processo de otimização toma um máximo de 20000 avaliações. Demais parâmetros são mostrados na Tabela 7.3.

Tabela 7.3: Parâmetros de treinamento para o GEMOPSO

Parâmetro	Valor
Rodadas por método	30
Rodadas MOPSO	10
Tam. População	100
Max Avaliações por método	20000
Max Avaliações por método	10000
Limite inferior	7 genes
Limite superior	20 genes
Cruzamento	Um Ponto, 90%
Mutação	Mutação Inteira, 1%
Duplicação	1%
Prune	1%

7.5 Resultados

Considerando os dados presentes nas Tabelas 7.4 à 7.19. Os valores representam os ranques médios, calculados, baseados nas 30 execuções individuais de cada algoritmo em cada problema do cenário para diferentes quantidades de objetivos. O ranque final (valor em parenteses) foi obtido baseado no teste estatístico de Kruskall e Wallis [28], e usado para representar os melhores algoritmos. Os algoritmos que tiveram melhores resultados com diferença estatística são mostrados em negrito com fundo cinza, e tendo o valor de 1.00 no melhor caso. Quando um algoritmo possui um ranque final diferente do melhor caso, significa que este algoritmo empatou com um ou vários outros. Nestes casos, o valor associado ao ranque final é a média da soma dos ranques finais dos algoritmos empatados. Posteriormente, a Tabela 7.20 mostra quais foram os componentes mais selecionados para ambos os algoritmos de projeto, focando na análise do impacto que eles tiveram na criação e otimização dos MOPSOs.

7.5.1 Comparação entre gramáticas

Os resultados para a comparação realizada entre as versões da Gramática estão presentes nas Tabelas 7.4, 7.5, 7.6 e 7.7, para a versão da gramática sem os recursos recursivos, e nas Tabelas 7.8, 7.9, 7.10 e 7.11, para a gramática que possui os recursos recursivos adicionados.

Tabela 7.4: Comparação entre gramáticas para a GE com 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	GE1 A	158.27 (5.50)	157.23 (5.00)	147.70 (5.00)	159.50 (5.50)	141.47 (5.00)	126.17 (5.00)	88.78 (3.00)
	GE2 A	108.07 (4.50)	124.53 (5.00)	136.70 (5.00)	91.90 (3.00)	145.03 (5.00)	145.40 (5.00)	79.03 (3.00)
	GE3 A	140.13 (5.00)	124.73 (5.00)	122.10 (5.00)	141.07 (5.50)	120.00 (5.00)	134.93 (5.00)	**56.30 (2.50)**
	GE1 B	**43.15 (2.00)**	**16.08 (1.50)**	60.50 (2.50)	54.90 (2.50)	**59.47 (2.00)**	**22.97 (1.50)**	117.23 (4.50)
	GE2 B	**44.63 (2.00)**	52.20 (2.00)	54.90 (2.00)	**30.97 (2.00)**	**33.17 (2.00)**	67.23 (2.50)	**64.17 (2.50)**
	GE3 B	**48.75 (2.00)**	68.22 (2.50)	**21.10 (1.50)**	64.67 (2.50)	**43.87 (2.00)**	46.30 (2.00)	137.48 (5.50)
3	GE1 A	145.07 (5.00)	138.03 (5.00)	115.57 (4.50)	117.37 (5.00)	126.87 (5.00)	126.30 (5.00)	65.92 (2.50)
	GE2 A	146.67 (5.00)	143.97 (5.00)	141.13 (5.00)	126.33 (5.00)	132.57 (5.00)	127.77 (5.00)	112.77 (4.50)
	GE3 A	114.77 (5.00)	124.50 (5.00)	146.80 (5.00)	115.80 (5.00)	147.07 (5.00)	152.40 (5.00)	94.20 (3.50)
	GE1 B	**34.83 (1.50)**	46.70 (2.00)	**16.50 (1.50)**	**74.47 (2.00)**	**40.20 (2.00)**	**41.90 (2.00)**	**46.13 (2.00)**
	GE2 B	**28.00 (1.50)**	**22.00 (1.50)**	44.50 (2.00)	**66.37 (2.00)**	**35.53 (2.00)**	**41.03 (2.00)**	72.45 (2.50)
	GE3 B	73.67 (3.00)	67.80 (2.50)	78.50 (3.00)	**42.67 (2.00)**	**60.77 (2.00)**	**53.60 (2.00)**	151.53 (6.00)
5	GE1 A	101.17 (3.50)	134.40 (5.50)	129.27 (5.00)	152.90 (5.50)	123.25 (5.00)	131.45 (5.00)	107.57 (5.00)
	GE2 A	146.73 (5.50)	164.13 (5.50)	**51.87 (2.00)**	148.00 (5.50)	144.07 (5.00)	80.52 (3.50)	143.00 (5.00)
	GE3 A	85.83 (3.50)	93.97 (3.00)	98.60 (4.50)	99.20 (3.50)	**57.72 (2.00)**	109.58 (4.00)	**60.53 (2.00)**
	GE1 B	**23.57 (1.00)**	59.13 (2.50)	**58.17 (2.00)**	**29.53 (1.50)**	**39.93 (2.00)**	**32.00 (1.50)**	**48.50 (2.00)**
	GE2 B	63.87 (3.00)	62.73 (2.50)	136.20 (5.00)	43.83 (2.00)	110.03 (5.00)	150.95 (5.50)	145.63 (5.00)
	GE3 B	121.83 (4.50)	**28.63 (2.00)**	68.90 (2.50)	69.53 (3.00)	**68.00 (2.00)**	**38.50 (1.50)**	**37.77 (2.00)**

Tabela 7.5: Comparação entre gramáticas para a GE com 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	GE1 A	149.07 (5.00)	136.53 (5.00)	71.43 (2.50)	142.27 (5.00)	152.33 (5.00)	148.70 (5.00)	136.00 (5.00)
	GE2 A	139.20 (5.00)	131.50 (5.00)	102.10 (3.50)	133.03 (5.00)	123.20 (5.00)	142.20 (5.00)	129.43 (5.00)
	GE3 A	118.23 (5.00)	138.47 (5.00)	**34.63 (1.50)**	131.20 (5.00)	127.53 (5.00)	115.60 (5.00)	122.10 (5.00)
	GE1 B	**15.57 (1.00)**	66.00 (2.50)	116.67 (5.00)	**33.85 (2.00)**	**49.87 (2.00)**	**42.03 (2.00)**	**60.67 (2.00)**
	GE2 B	55.10 (2.50)	55.00 (2.50)	74.98 (3.00)	**32.68 (2.00)**	**42.10 (2.00)**	**44.47 (2.00)**	**41.67 (2.00)**
	GE3 B	65.83 (2.50)	**15.50 (1.00)**	143.18 (5.50)	**69.97 (2.00)**	**47.97 (2.00)**	**50.00 (2.00)**	**53.13 (2.00)**
3	GE1 A	134.57 (5.00)	118.23 (5.00)	97.27 (4.00)	141.67 (5.00)	143.53 (5.00)	112.60 (4.50)	101.73 (4.00)
	GE2 A	141.13 (5.00)	138.67 (5.00)	105.40 (4.00)	148.00 (5.00)	147.10 (5.00)	131.73 (4.50)	126.47 (5.00)
	GE3 A	130.70 (5.00)	149.60 (5.00)	**21.03 (1.50)**	116.83 (5.00)	115.37 (5.00)	137.17 (5.00)	73.57 (3.00)
	GE1 B	**21.17 (1.50)**	**21.40 (1.50)**	**49.57 (1.50)**	39.12 (2.00)	48.13 (2.00)	**18.03 (1.50)**	**54.60 (2.00)**
	GE2 B	44.37 (2.00)	52.70 (2.00)	117.60 (4.50)	**26.18 (1.50)**	**18.97 (1.50)**	96.40 (4.00)	110.10 (4.00)
	GE3 B	71.07 (2.50)	62.40 (2.50)	152.13 (5.50)	71.20 (2.50)	69.90 (2.50)	**47.07 (1.50)**	76.53 (3.00)
5	GE1 A	68.87 (3.00)	137.67 (5.00)	105.43 (4.00)	132.20 (5.50)	138.37 (5.50)	116.43 (4.50)	119.10 (4.00)
	GE2 A	98.03 (3.00)	89.52 (3.50)	124.77 (5.00)	163.47 (5.50)	162.57 (5.50)	120.17 (4.50)	159.07 (6.00)
	GE3 A	**22.17 (1.00)**	103.82 (4.00)	**25.60 (1.50)**	93.23 (3.50)	74.23 (3.00)	92.97 (4.00)	97.13 (4.00)
	GE1 B	138.37 (5.50)	**33.48 (1.50)**	47.10 (2.00)	**39.83 (2.00)**	**34.53 (1.50)**	**31.33 (1.50)**	**35.20 (1.50)**
	GE2 B	72.77 (3.00)	147.07 (5.50)	159.30 (5.50)	68.53 (2.50)	57.43 (2.50)	144.40 (5.00)	86.60 (4.00)
	GE3 B	142.80 (5.50)	**31.45 (1.50)**	80.80 (3.00)	**45.73 (2.00)**	75.87 (3.00)	**37.70 (1.50)**	**45.90 (1.50)**

As Tabelas 7.4 e 7.5 apresentam os resultados da comparação entre as gramáticas não recursivas, para os MOPSOs gerados com a GE para 20 e 60 variáveis. Na maioria dos casos, independente de problemas ou número de variáveis, a gramática B demonstra obter melhores resultados em comparação com a gramática A. Nos cenários que consideraram 20 variáveis houve mais empates entre os algoritmos do que os cenários para 60 variáveis. Os empates indicam que os algoritmos apresentaram comportamento semelhante no problema testado, obtendo resultados próximos.

A comparação entre as gramáticas considerando o algoritmo IRACE para 20 e 60 variáveis está presente nas Tabelas 7.6 e 7.7. Diferente da comparação para a GE, o IRACE demonstra gerar MOPSOs com melhor desempenho quando utilizando a gramática A, porém, a diferença entre as gramáticas não é tão grande, considerando a quantidade de problemas em que

Tabela 7.6: Comparação entre gramáticas para o IRACE com 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	IR1 A	93.43 (3.50)	101.17 (4.00)	**43.50 (1.50)**	129.10 (5.00)	86.37 (3.50)	119.17 (4.00)	**62.23 (2.00)**
	IR2 A	98.90 (3.50)	**32.80 (1.50)**	61.57 (2.00)	**59.60 (2.00)**	97.30 (3.50)	89.00 (4.00)	126.23 (5.00)
	IR3 A	75.40 (3.50)	125.13 (4.50)	114.33 (4.50)	**64.50 (2.00)**	**82.10 (3.00)**	89.53 (4.00)	129.90 (5.00)
	IR1 B	101.10 (4.00)	126.53 (4.50)	84.47 (3.50)	**57.73 (2.00)**	87.03 (3.50)	**41.90 (1.00)**	**40.70 (2.00)**
	IR2 B	112.50 (4.00)	65.63 (2.50)	112.33 (4.50)	128.97 (5.00)	124.13 (4.50)	112.50 (4.00)	111.93 (5.00)
	IR3 B	**61.67 (2.50)**	91.73 (4.00)	126.80 (5.00)	103.10 (5.00)	**66.07 (3.00)**	90.90 (4.00)	**72.00 (2.00)**
3	IR1 A	127.80 (5.00)	87.90 (3.50)	**63.90 (2.50)**	57.73 (2.50)	119.03 (5.00)	106.37 (4.50)	**69.23 (2.00)**
	IR2 A	67.40 (2.50)	96.43 (3.50)	114.43 (4.50)	94.73 (3.00)	119.27 (5.00)	107.43 (4.50)	**55.53 (2.00)**
	IR3 A	70.37 (2.50)	**60.87 (2.50)**	**67.83 (2.50)**	**38.20 (1.50)**	**54.03 (2.00)**	60.80 (2.00)	113.00 (5.00)
	IR1 B	101.10 (4.00)	100.23 (4.00)	77.20 (3.00)	135.30 (5.50)	120.67 (5.00)	**22.33 (1.00)**	**66.77 (2.00)**
	IR2 B	125.03 (5.00)	117.03 (4.00)	86.00 (3.00)	83.07 (3.00)	**59.13 (2.00)**	107.50 (4.50)	116.63 (5.00)
	IR3 B	**51.30 (2.00)**	80.53 (3.50)	133.63 (5.50)	133.97 (5.50)	**70.87 (2.00)**	138.57 (4.50)	121.83 (5.00)
5	IR1 A	91.87 (3.00)	**55.37 (2.00)**	**66.57 (2.50)**	115.60 (4.50)	111.97 (4.00)	111.47 (4.50)	67.37 (3.00)
	IR2 A	**61.47 (2.50)**	122.97 (5.00)	93.47 (3.00)	**37.80 (1.50)**	86.87 (3.50)	146.50 (5.50)	102.90 (4.00)
	IR3 A	103.97 (4.50)	**52.90 (2.00)**	**61.07 (2.50)**	121.57 (4.50)	80.50 (3.50)	86.50 (3.50)	99.13 (4.00)
	IR1 B	**59.47 (2.50)**	120.30 (5.00)	136.57 (5.50)	**48.47 (1.50)**	97.37 (3.50)	102.27 (3.50)	**42.40 (1.50)**
	IR2 B	139.30 (5.50)	79.27 (2.50)	119.73 (5.00)	117.37 (4.50)	**71.17 (3.00)**	**32.03 (1.50)**	96.53 (4.00)
	IR3 B	86.93 (3.00)	112.20 (4.50)	**65.60 (2.50)**	102.20 (4.50)	95.13 (3.50)	64.23 (2.50)	134.67 (4.50)

Tabela 7.7: Comparação entre gramáticas para o IRACE com 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	IR1 A	80.07 (3.50)	109.13 (4.50)	**74.53 (3.50)**	85.83 (3.50)	66.10 (2.50)	75.70 (3.00)	97.17 (3.50)
	IR2 A	93.17 (3.50)	100.10 (4.50)	**94.10 (3.50)**	107.63 (4.00)	105.87 (4.50)	68.97 (2.50)	138.50 (5.50)
	IR3 A	116.03 (4.00)	**45.33 (1.50)**	**88.30 (3.50)**	99.83 (3.50)	89.23 (3.50)	108.37 (4.50)	65.90 (2.50)
	IR1 B	**55.93 (2.00)**	116.93 (4.50)	**91.77 (3.50)**	99.30 (3.50)	**62.03 (2.00)**	123.47 (5.00)	66.17 (2.50)
	IR2 B	98.17 (4.00)	**48.90 (1.50)**	**83.50 (3.50)**	87.37 (3.50)	101.97 (4.00)	104.70 (4.00)	**51.80 (2.00)**
	IR3 B	99.63 (4.00)	122.60 (4.50)	**110.80 (3.50)**	**63.03 (3.00)**	117.80 (4.50)	**61.80 (2.00)**	123.47 (5.00)
3	IR1 A	**84.00 (3.50)**	105.33 (4.50)	**69.33 (3.00)**	83.77 (3.00)	119.90 (4.50)	111.40 (5.00)	**44.80 (1.00)**
	IR2 A	**97.10 (3.50)**	**28.13 (1.50)**	**77.93 (3.00)**	**52.73 (2.50)**	**49.47 (1.50)**	72.53 (2.50)	110.97 (4.00)
	IR3 A	**75.87 (3.50)**	133.60 (5.00)	127.13 (5.50)	**71.17 (2.50)**	138.70 (5.50)	**28.73 (1.00)**	98.03 (4.00)
	IR1 B	**75.77 (3.50)**	81.50 (3.00)	107.10 (3.50)	127.50 (5.50)	91.10 (3.50)	114.77 (5.00)	94.47 (4.00)
	IR2 B	**113.10 (3.50)**	134.53 (5.00)	**76.90 (3.00)**	88.53 (3.00)	89.60 (3.50)	71.93 (2.50)	92.50 (4.00)
	IR3 B	**97.17 (3.50)**	59.90 (2.00)	**84.60 (3.00)**	119.30 (4.50)	54.23 (2.50)	143.63 (5.00)	102.23 (4.00)
5	IR1 A	**70.43 (2.50)**	95.20 (4.00)	**64.00 (2.50)**	81.73 (3.00)	**64.77 (2.00)**	**74.90 (3.00)**	140.80 (6.00)
	IR2 A	98.20 (3.50)	102.93 (4.00)	75.17 (3.00)	81.00 (3.00)	**56.20 (2.00)**	114.90 (4.50)	**85.50 (3.00)**
	IR3 A	118.03 (5.00)	79.90 (3.00)	105.03 (4.00)	134.07 (5.50)	115.13 (4.50)	87.37 (3.50)	**65.67 (3.00)**
	IR1 B	**65.57 (2.50)**	123.40 (4.50)	76.00 (3.00)	102.63 (4.00)	108.07 (4.50)	**68.53 (3.00)**	**80.90 (3.00)**
	IR2 B	121.93 (5.00)	92.07 (4.00)	129.83 (5.00)	**59.83 (2.50)**	117.77 (4.50)	102.10 (3.50)	**84.87 (3.00)**
	IR3 B	**68.83 (2.50)**	**49.50 (1.50)**	92.97 (3.50)	83.73 (3.00)	81.07 (3.50)	95.20 (3.50)	**85.27 (3.00)**

as gramáticas A ou B apresentam melhores resultados. Mesmo quando ocorrem empates entre os algoritmos, os empates ocorrem para ambas as gramáticas na maioria das vezes.

Tabela 7.8: Comparação entre gramáticas recursivas para a GE com 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	GE1 A	107.77 (4.50)	111.17 (4.50)	138.17 (5.00)	153.40 (5.50)	137.30 (5.00)	118.57 (5.00)	112.83 (4.50)
	GE2 A	134.90 (5.00)	127.83 (5.00)	135.33 (5.00)	105.50 (4.00)	131.47 (5.00)	154.77 (5.00)	127.67 (5.00)
	GE3 A	163.83 (5.50)	164.70 (5.50)	133.00 (5.00)	147.60 (5.50)	137.73 (5.00)	133.17 (5.00)	165.50 (5.50)
	GE1 B	**44.70 (2.00)**	**48.02 (2.00)**	**32.18 (1.50)**	54.07 (2.00)	**33.55 (1.50)**	**32.43 (1.50)**	**17.93 (1.50)**
	GE2 B	**50.30 (2.00)**	**50.78 (2.00)**	71.87 (3.00)	**21.30 (1.50)**	73.57 (3.00)	74.57 (3.00)	45.93 (2.00)
	GE3 B	**41.50 (2.00)**	**40.50 (2.00)**	**32.45 (1.50)**	61.13 (2.50)	**29.38 (1.50)**	**29.50 (1.50)**	73.13 (2.50)
3	GE1 A	69.80 (3.00)	63.67 (2.50)	115.87 (4.50)	57.33 (2.50)	108.57 (4.50)	104.10 (4.50)	77.53 (3.00)
	GE2 A	136.27 (5.00)	137.17 (5.00)	124.57 (4.50)	135.77 (5.50)	132.43 (5.00)	126.00 (4.50)	123.90 (4.50)
	GE3 A	164.73 (5.50)	163.83 (5.50)	165.50 (6.00)	165.23 (5.50)	165.50 (5.50)	165.50 (6.00)	165.13 (6.00)
	GE1 B	**28.27 (1.50)**	**19.90 (1.50)**	60.70 (2.50)	**31.30 (2.00)**	61.97 (2.50)	**44.87 (2.00)**	50.43 (2.00)
	GE2 B	105.47 (4.00)	103.43 (4.50)	**16.45 (1.00)**	94.33 (3.00)	**21.97 (1.50)**	**60.20 (2.00)**	**22.90 (1.50)**
	GE3 B	38.47 (2.00)	55.00 (2.00)	59.92 (2.50)	59.03 (2.50)	52.57 (2.00)	**42.33 (2.00)**	103.10 (4.00)
	GE1 A	86.30 (3.50)	102.17 (4.00)	155.07 (5.50)	96.40 (4.00)	136.30 (5.50)	148.37 (5.00)	159.57 (5.50)
	GE2 A	79.73 (3.50)	87.27 (3.50)	**23.40 (2.00)**	93.13 (4.00)	**45.87 (2.50)**	**44.00 (2.00)**	**62.20 (2.00)**
	GE3 A	108.47 (3.50)	128.93 (5.00)	106.67 (4.50)	110.27 (4.00)	148.65 (5.50)	134.10 (5.00)	114.20 (4.50)
	GE1 B	90.23 (3.50)	78.53 (3.00)	**55.10 (2.00)**	106.77 (4.00)	**65.63 (2.50)**	**47.83 (2.00)**	**31.97 (2.00)**
	GE2 B	**16.87 (1.00)**	**21.37 (1.00)**	144.17 (5.00)	**27.40 (1.00)**	**76.70 (2.50)**	117.30 (5.00)	128.00 (5.00)
	GE3 B	161.40 (6.00)	124.73 (4.50)	**58.60 (2.00)**	109.03 (4.00)	**69.85 (2.50)**	**51.40 (2.00)**	**47.07 (2.00)**

Tabela 7.9: Comparação entre gramáticas recursivas para a GE com 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	GE1 A	135.53 (5.00)	143.93 (5.00)	110.37 (4.50)	106.73 (4.50)	106.70 (4.00)	142.93 (5.00)	126.93 (5.00)
	GE2 A	141.80 (5.00)	131.90 (5.00)	131.03 (5.00)	134.27 (5.00)	134.30 (5.00)	141.83 (5.00)	143.23 (5.00)
	GE3 A	123.17 (5.00)	124.73 (5.00)	164.63 (5.50)	165.50 (5.50)	165.50 (5.50)	118.73 (5.00)	136.33 (5.00)
	GE1 B	**58.07 (2.00)**	**33.77 (1.50)**	**36.50 (2.00)**	**36.83 (2.00)**	**33.27 (1.50)**	**34.52 (1.50)**	50.63 (2.00)
	GE2 B	**27.40 (2.00)**	77.17 (3.00)	**64.50 (2.00)**	**64.67 (2.00)**	74.23 (3.50)	73.60 (3.00)	**18.60 (1.50)**
	GE3 B	**57.03 (2.00)**	**31.50 (1.50)**	**35.97 (2.00)**	**35.00 (2.00)**	**29.00 (1.50)**	**31.38 (1.50)**	67.27 (2.50)
3	GE1 A	125.63 (4.50)	117.07 (4.50)	**62.40 (2.00)**	71.90 (3.00)	71.33 (3.00)	112.63 (4.00)	**63.40 (2.50)**
	GE2 A	116.33 (4.50)	123.43 (4.50)	107.80 (4.50)	135.50 (5.00)	135.73 (5.00)	114.63 (4.00)	152.90 (5.50)
	GE3 A	164.53 (6.00)	165.50 (6.00)	161.77 (6.00)	165.50 (5.50)	165.27 (5.50)	164.50 (6.00)	148.10 (5.50)
	GE1 B	**55.03 (2.00)**	**48.68 (2.00)**	**44.67 (2.00)**	**15.50 (1.50)**	**17.57 (1.50)**	47.50 (2.00)	**66.63 (2.50)**
	GE2 B	**21.90 (2.00)**	**36.60 (2.00)**	115.10 (4.50)	103.67 (4.00)	103.23 (4.00)	**22.07 (1.50)**	**44.63 (2.50)**
	GE3 B	**59.57 (2.00)**	**51.72 (2.00)**	**51.27 (2.00)**	50.93 (2.00)	49.87 (2.00)	81.67 (3.50)	**67.33 (2.50)**
5	GE1 A	124.70 (5.00)	119.58 (5.00)	144.07 (5.50)	79.73 (3.00)	65.37 (3.00)	153.50 (5.50)	119.13 (5.00)
	GE2 A	**21.13 (1.50)**	76.90 (2.50)	**17.43 (1.50)**	62.77 (2.50)	92.30 (3.50)	**61.73 (2.00)**	**69.90 (2.00)**
	GE3 A	128.07 (5.00)	132.43 (5.00)	105.13 (4.00)	129.43 (5.00)	147.13 (5.50)	113.27 (4.50)	145.37 (5.00)
	GE1 B	82.63 (2.50)	**32.50 (1.50)**	66.23 (2.50)	101.03 (3.50)	126.83 (4.50)	**23.57 (2.00)**	**61.13 (2.00)**
	GE2 B	47.53 (2.00)	137.75 (5.00)	156.93 (5.50)	**29.60 (1.50)**	**20.47 (1.00)**	139.70 (5.00)	111.33 (5.00)
	GE3 B	138.93 (5.00)	43.83 (2.00)	53.20 (2.00)	140.43 (5.50)	90.90 (3.50)	**51.23 (2.00)**	**36.13 (2.00)**

Considerando as gramáticas que incluem as regras recursivas para a criação dos MOPSOs, os resultados para a comparação entre gramáticas recursivas para a GE com 20 variáveis estão representados nas Tabelas 7.8 e 7.9. Aqui, os resultados se mostraram semelhantes aos obtidos com a gramática não recursiva, onde os MOPSOs gerados com a gramática B demonstraram melhor desempenho tanto para 20 quanto para 60 variáveis. Dos poucos casos onde a gramática A obteve bons resultados, a maioria foi para os problemas considerando 5 objetivos, ou seja, instâncias de problemas mais complexos.

Tabela 7.10: Comparação entre gramáticas recursivas para o IRACE com 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	IR1 A	**69.10 (3.00)**	126.93 (4.50)	112.20 (4.50)	125.90 (5.00)	132.30 (5.50)	104.40 (4.00)	74.90 (3.00)
	IR2 A	103.57 (3.50)	**65.67 (2.50)**	125.47 (5.00)	142.97 (5.50)	**89.13 (3.00)**	118.80 (4.00)	132.20 (5.50)
	IR3 A	123.77 (5.00)	104.30 (4.50)	80.10 (2.50)	78.70 (2.50)	93.97 (3.50)	95.13 (4.00)	157.43 (5.50)
	IR1 B	**79.63 (3.00)**	88.93 (3.50)	**46.90 (2.00)**	**45.83 (2.00)**	**66.77 (3.00)**	**37.00 (1.00)**	**23.50 (1.00)**
	IR2 B	85.63 (3.50)	91.60 (3.50)	**57.83 (2.00)**	**48.53 (2.00)**	**92.50 (3.00)**	99.67 (4.00)	72.27 (3.00)
	IR3 B	**81.30 (3.00)**	**65.57 (2.50)**	120.50 (5.00)	101.07 (4.00)	**68.33 (3.00)**	88.00 (4.00)	82.70 (3.00)
3	IR1 A	133.83 (5.00)	89.90 (3.50)	65.43 (2.00)	**22.63 (1.00)**	104.67 (4.00)	80.63 (3.00)	**48.37 (1.50)**
	IR2 A	**33.13 (1.00)**	115.77 (4.00)	107.87 (4.50)	130.63 (5.00)	77.13 (3.50)	142.90 (5.50)	105.33 (4.00)
	IR3 A	112.83 (4.00)	102.03 (4.00)	**33.30 (1.50)**	78.17 (3.00)	96.33 (3.50)	**48.50 (2.00)**	92.87 (4.00)
	IR1 B	80.87 (3.50)	**63.53 (2.50)**	130.93 (5.00)	74.97 (3.00)	94.53 (3.50)	109.50 (4.50)	85.37 (3.50)
	IR2 B	106.20 (4.00)	88.47 (3.50)	116.30 (4.50)	112.10 (4.00)	104.47 (4.00)	**59.10 (2.00)**	113.13 (4.00)
	IR3 B	76.13 (3.50)	83.30 (3.50)	89.17 (3.50)	124.50 (5.00)	**65.87 (2.50)**	102.37 (4.00)	97.93 (4.00)
5	IR1 A	143.20 (5.50)	85.50 (3.50)	147.53 (5.50)	133.07 (5.00)	144.17 (5.50)	134.43 (5.00)	146.57 (5.50)
	IR2 A	**47.13 (2.00)**	**68.93 (3.00)**	**52.50 (2.00)**	98.40 (4.00)	**76.90 (2.50)**	**29.20 (1.50)**	**44.97 (2.50)**
	IR3 A	146.47 (5.50)	95.13 (3.50)	126.00 (5.00)	127.50 (5.00)	133.10 (5.50)	126.87 (5.00)	143.10 (5.50)
	IR1 B	**51.57 (2.00)**	81.83 (3.50)	92.07 (4.00)	67.50 (2.50)	**64.43 (2.50)**	73.60 (2.50)	**74.27 (2.50)**
	IR2 B	91.47 (3.50)	101.07 (3.50)	77.37 (2.50)	**32.80 (1.50)**	**52.63 (2.50)**	66.90 (2.00)	**71.57 (2.50)**
	IR3 B	63.17 (2.50)	110.53 (4.00)	**47.53 (2.00)**	83.73 (3.00)	**71.77 (2.50)**	112.00 (5.00)	**62.53 (2.50)**

Por fim, as Tabelas 7.10 e 7.11 apresentam os resultados da comparação entre as gramáticas recursivas para o IRACE com 20 e 60 variáveis. Dentre as comparações feitas até então, esta se mostrou mais equilibrada, onde apesar da gramática B ter proporcionado que seus MOPSOs obtivessem melhores resultados em mais situações, a quantidade de vezes que isso acontece é levemente maior que a quantidade de vezes que a gramática A obtém bons resultados. Principalmente para os problemas que consideravam 3 objetivos para ambas 20 e 60 variáveis, ambas as gramáticas obtiveram os resultados semelhantes e com poucos empates.

A conclusão obtida é que a gramática B, que inclui modificações em algumas de suas regras, proporcionou a criação de MOPSOs melhores em relação aos criados utilizando a gramática A. Dentre as comparações realizadas, a única que não seguiu um padrão foi na

Tabela 7.11: Comparação entre gramáticas recursivas para o IRACE com 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	DTLZ7
2	IR1 A	91.17 (3.50)	89.27 (3.50)	72.07 (2.50)	131.73 (5.00)	109.10 (4.50)	109.83 (4.50)	114.13 (4.50)
	IR2 A	121.33 (5.00)	89.73 (3.50)	**51.00 (2.00)**	123.70 (5.00)	115.40 (4.50)	135.60 (5.00)	138.60 (5.00)
	IR3 A	135.70 (5.50)	113.90 (4.00)	138.10 (5.50)	**61.53 (2.50)**	126.77 (4.50)	50.33 (2.00)	122.10 (5.00)
	IR1 B	78.53 (3.00)	78.30 (3.50)	91.07 (3.50)	**69.87 (2.50)**	**35.77 (1.50)**	**31.60 (1.50)**	**49.40 (2.00)**
	IR2 B	**36.93 (1.00)**	**61.00 (2.50)**	114.27 (4.50)	**62.17 (2.50)**	**52.57 (1.50)**	87.50 (3.00)	78.50 (2.50)
	IR3 B	79.33 (3.00)	110.80 (4.00)	76.50 (3.00)	94.00 (3.50)	103.40 (4.50)	128.13 (5.00)	**40.27 (2.00)**
3	IR1 A	73.87 (2.50)	115.00 (4.50)	**62.00 (2.00)**	108.30 (4.00)	105.23 (4.00)	119.17 (5.00)	137.90 (5.50)
	IR2 A	**51.13 (2.00)**	92.63 (3.50)	109.57 (4.50)	91.43 (3.50)	98.27 (4.00)	**33.30 (1.50)**	112.70 (4.00)
	IR3 A	123.60 (5.00)	132.87 (5.50)	109.47 (4.50)	100.07 (4.00)	**40.60 (1.00)**	62.37 (2.00)	79.90 (3.00)
	IR1 B	106.97 (4.00)	**49.83 (1.50)**	**53.63 (2.00)**	96.30 (4.00)	115.97 (4.00)	127.97 (5.00)	85.63 (3.00)
	IR2 B	116.37 (5.00)	93.23 (3.50)	120.53 (4.50)	91.17 (3.50)	87.70 (4.00)	79.03 (2.50)	**50.50 (2.50)**
	IR3 B	71.07 (2.50)	59.43 (2.50)	87.80 (3.50)	**55.73 (2.00)**	95.23 (4.00)	121.17 (5.00)	76.37 (3.00)
5	IR1 A	148.53 (5.50)	111.97 (4.00)	153.33 (5.50)	146.97 (5.50)	91.67 (3.50)	152.07 (5.50)	141.13 (5.50)
	IR2 A	**81.30 (2.50)**	73.30 (2.50)	50.40 (2.50)	**46.10 (2.50)**	**63.87 (3.00)**	97.80 (4.00)	64.47 (3.00)
	IR3 A	139.77 (5.50)	153.97 (6.00)	143.83 (5.50)	150.33 (5.50)	107.93 (4.00)	133.57 (5.00)	135.53 (5.00)
	IR1 B	**66.80 (2.50)**	**41.03 (2.00)**	**33.97 (1.50)**	**65.10 (2.50)**	82.70 (3.50)	**45.53 (2.00)**	77.23 (3.00)
	IR2 B	**50.67 (2.50)**	76.77 (3.00)	85.43 (3.00)	**71.77 (2.50)**	100.10 (3.50)	**42.27 (2.00)**	**25.63 (1.00)**
	IR3 B	**55.93 (2.50)**	85.97 (3.50)	76.03 (3.00)	**62.73 (2.50)**	96.73 (3.50)	71.77 (2.50)	99.00 (3.50)

comparação entre as gramáticas não recursivas para o IRACE, onde apesar de muito próximos, os resultados indicaram que a gramática A foi superior. Nos demais casos, a gramática B sempre obteve vantagem para a maioria dos problemas.

7.5.2 Comparação entre algoritmos de projeto

Após realizar a comparação entre as gramáticas e constatar que a gramática modificada obtém melhores resultados, os algoritmos GE e IRACE são avaliados. Os resultados obtidos com a experimentação e avaliação destes algoritmos, estão presentes nas Tabelas 7.12 e 7.13 para a gramática não recursiva, e Tabelas 7.14 e 7.15 para a gramática recursiva.

Tabela 7.12: Comparação entre GE e IRACE para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	131.50 (5.00)	140.07 (5.00)	145.70 (5.00)	147.60 (5.00)	138.43 (5.00)	147.40 (5.00)
	IR2	137.30 (5.00)	137.47 (5.00)	126.07 (5.00)	136.03 (5.00)	122.17 (5.00)	134.23 (5.00)
	IR3	137.70 (5.00)	128.97 (5.00)	134.73 (5.00)	122.87 (5.00)	145.90 (5.00)	124.87 (5.00)
	GE1	**43.05 (2.00)**	**42.55 (2.00)**	54.63 (2.00)	**36.75 (2.00)**	**45.53 (2.00)**	49.23 (2.00)
	GE2	**41.95 (2.00)**	**31.57 (2.00)**	**20.23 (1.50)**	**50.65 (2.00)**	**49.53 (2.00)**	**18.17 (1.50)**
	GE3	**51.50 (2.00)**	**62.38 (2.00)**	61.63 (2.50)	**49.10 (2.00)**	**41.43 (2.00)**	69.10 (2.50)
3	IR1	113.27 (5.00)	122.70 (5.00)	135.33 (5.00)	130.70 (5.00)	158.10 (5.00)	133.80 (5.00)
	IR2	150.67 (5.00)	135.43 (5.00)	141.97 (5.00)	141.80 (5.00)	120.57 (5.00)	129.57 (5.00)
	IR3	142.57 (5.00)	148.37 (5.00)	129.20 (5.00)	134.00 (5.00)	127.83 (5.00)	143.13 (5.00)
	GE1	69.03 (2.50)	**52.72 (2.00)**	**31.70 (2.00)**	**50.10 (2.00)**	**44.40 (2.00)**	**36.47 (2.00)**
	GE2	40.12 (2.00)	**36.10 (2.00)**	**69.40 (2.00)**	**59.73 (2.00)**	**45.87 (2.00)**	**40.70 (2.00)**
	GE3	**27.35 (1.50)**	**47.68 (2.00)**	**35.40 (2.00)**	**26.67 (2.00)**	**46.23 (2.00)**	**59.33 (2.00)**
5	IR1	158.73 (5.00)	140.70 (5.00)	130.77 (5.00)	129.37 (5.00)	155.57 (5.50)	137.43 (5.00)
	IR2	126.83 (5.00)	132.37 (5.00)	150.87 (5.00)	130.10 (5.00)	138.87 (5.00)	145.03 (5.00)
	IR3	120.93 (5.00)	133.43 (5.00)	124.87 (5.00)	147.03 (5.00)	112.07 (4.50)	124.03 (5.00)
	GE1	**68.33 (2.00)**	**48.15 (2.00)**	**60.57 (2.00)**	**50.68 (2.00)**	**45.78 (2.00)**	**54.07 (2.00)**
	GE2	**34.73 (2.00)**	**41.53 (2.00)**	**31.90 (2.00)**	**32.85 (2.00)**	**60.98 (2.00)**	**33.27 (2.00)**
	GE3	**33.43 (2.00)**	**46.82 (2.00)**	**44.03 (2.00)**	**52.97 (2.00)**	**29.73 (2.00)**	**49.17 (2.00)**

Nas Tabelas 7.12 e 7.13, em todas as instâncias consideradas na comparação entre os algoritmos de geração de MOPSOs, a GE obteve vantagem em todos os cenários, sejam eles para 20 ou 60 variáveis. Dentre todos os casos avaliados, não houve situações onde os MOPSOs gerados pelo IRACE obtiveram resultados estatisticamente melhores de acordo com o ranqueamento utilizado na comparação. Com exceção de alguns poucos casos, quase todos os MOPSOs gerados pela GE empataram entre si, demonstrando um comportamento semelhante na obtenção dos resultados.

Tabela 7.13: Comparação entre GE e IRACE para 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	132.47 (5.00)	123.00 (5.00)	99.50 (4.00)	150.83 (5.00)	128.03 (5.00)	137.20 (5.00)
	IR2	117.67 (5.00)	138.50 (5.00)	117.03 (4.50)	124.97 (5.00)	124.97 (5.00)	136.80 (5.00)
	IR3	148.10 (5.00)	145.00 (5.00)	143.03 (5.00)	130.70 (5.00)	153.50 (5.00)	132.50 (5.00)
	GE1	62.43 (2.50)	**43.83 (2.00)**	**36.20 (1.50)**	**45.90 (2.00)**	48.43 (2.00)	**34.95 (2.00)**
	GE2	**18.93 (1.00)**	**38.07 (2.00)**	**30.53 (1.50)**	**30.43 (2.00)**	**23.30 (1.50)**	**41.65 (2.00)**
	GE3	63.40 (2.50)	**54.60 (2.00)**	116.70 (4.50)	**60.17 (2.00)**	64.77 (2.50)	**59.90 (2.00)**
3	IR1	127.63 (5.00)	116.43 (4.50)	140.13 (5.00)	138.57 (5.00)	132.60 (5.00)	121.83 (5.00)
	IR2	138.77 (5.00)	156.17 (5.50)	152.23 (5.00)	156.23 (5.50)	140.43 (5.00)	127.80 (5.00)
	IR3	140.10 (5.00)	133.90 (5.00)	114.13 (5.00)	111.70 (4.50)	133.47 (5.00)	156.87 (5.00)
	GE1	**40.33 (2.00)**	**57.83 (2.00)**	**41.53 (2.00)**	**43.67 (2.00)**	**57.33 (2.00)**	**42.10 (2.00)**
	GE2	**61.77 (2.00)**	**24.17 (2.00)**	**48.97 (2.00)**	**41.93 (2.00)**	**34.10 (2.00)**	**59.10 (2.00)**
	GE3	**34.40 (2.00)**	**54.50 (2.00)**	**46.00 (2.00)**	**50.90 (2.00)**	**45.07 (2.00)**	**35.30 (2.00)**
5	IR1	151.33 (5.00)	120.67 (5.00)	136.70 (5.00)	152.43 (5.00)	144.37 (5.00)	137.93 (5.00)
	IR2	134.97 (5.00)	132.13 (5.00)	115.47 (4.50)	120.00 (5.00)	140.10 (5.00)	122.40 (5.00)
	IR3	120.20 (5.00)	153.70 (5.00)	154.33 (5.50)	134.07 (5.00)	122.03 (5.00)	146.17 (5.00)
	GE1	**45.98 (2.00)**	**45.50 (2.00)**	**51.13 (2.00)**	**43.00 (2.00)**	**44.00 (2.00)**	**43.27 (2.00)**
	GE2	**44.50 (2.00)**	**45.50 (2.00)**	**54.73 (2.00)**	**43.00 (2.00)**	**44.00 (2.00)**	**62.37 (2.00)**
	GE3	**46.02 (2.00)**	**45.50 (2.00)**	**30.63 (2.00)**	**50.50 (2.00)**	**48.50 (2.00)**	**30.87 (2.00)**

Tabela 7.14: Comparação entre GE e IRACE recursivos para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	137.33 (5.00)	151.90 (5.00)	150.63 (5.00)	132.00 (5.00)	140.80 (5.00)	138.53 (5.00)
	IR2	126.67 (5.00)	133.97 (5.00)	132.10 (5.00)	124.63 (5.00)	130.43 (5.00)	127.87 (5.00)
	IR3	142.50 (5.00)	120.63 (5.00)	123.77 (5.00)	149.87 (5.00)	135.27 (5.00)	140.10 (5.00)
	GE1	**62.50 (2.00)**	**45.55 (2.00)**	**25.97 (2.00)**	58.50 (2.50)	67.17 (2.50)	**32.50 (1.50)**
	GE2	**26.33 (2.00)**	**34.50 (2.00)**	**63.63 (2.00)**	**16.50 (1.00)**	**17.17 (1.50)**	**30.98 (1.50)**
	GE3	**47.67 (2.00)**	**56.45 (2.00)**	**46.90 (2.00)**	61.50 (2.50)	52.17 (2.00)	73.02 (3.00)
3	IR1	136.77 (5.00)	134.30 (5.00)	134.37 (5.00)	138.60 (5.00)	137.47 (5.00)	124.43 (5.00)
	IR2	147.57 (5.00)	140.60 (5.00)	139.90 (5.00)	115.97 (5.00)	122.33 (5.00)	128.17 (5.00)
	IR3	122.17 (5.00)	131.60 (5.00)	132.23 (5.00)	151.93 (5.00)	146.70 (5.00)	152.33 (5.00)
	GE1	**37.07 (2.00)**	**37.08 (2.00)**	**17.97 (1.00)**	**55.17 (2.00)**	**51.98 (2.00)**	**15.50 (1.00)**
	GE2	**50.37 (2.00)**	**52.42 (2.00)**	56.43 (2.50)	**43.43 (2.00)**	**44.05 (2.00)**	62.77 (2.50)
	GE3	**49.07 (2.00)**	**47.00 (2.00)**	62.10 (2.50)	**37.90 (2.00)**	**40.47 (2.00)**	59.80 (2.50)
5	IR1	143.07 (5.00)	149.83 (5.00)	130.93 (5.00)	132.10 (5.00)	134.07 (5.00)	137.73 (5.00)
	IR2	143.47 (5.00)	138.23 (5.00)	138.13 (5.00)	138.00 (5.00)	136.40 (5.00)	139.00 (5.00)
	IR3	119.97 (5.00)	118.43 (5.00)	137.43 (5.00)	136.40 (5.00)	136.03 (5.00)	129.77 (5.00)
	GE1	**31.70 (2.00)**	**24.08 (2.00)**	**21.57 (1.50)**	**20.70 (1.50)**	**20.65 (1.50)**	37.73 (2.00)
	GE2	**65.90 (2.00)**	**61.53 (2.00)**	75.07 (2.50)	67.43 (2.50)	65.67 (2.50)	68.87 (2.50)
	GE3	**38.90 (2.00)**	**50.88 (2.00)**	39.87 (2.00)	48.37 (2.00)	50.18 (2.00)	**29.90 (1.50)**

Tabela 7.15: Comparação entre GE e IRACE recursivos para 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	136.63 (5.00)	137.70 (5.00)	130.53 (5.00)	142.07 (5.00)	135.37 (5.00)	145.67 (5.00)
	IR2	154.97 (5.50)	119.07 (5.00)	127.47 (5.00)	123.13 (5.00)	122.03 (5.00)	144.40 (5.00)
	IR3	114.90 (4.50)	149.73 (5.00)	148.50 (5.00)	141.30 (5.00)	149.10 (5.00)	116.43 (5.00)
	GE1	**29.12 (2.00)**	**39.28 (2.00)**	42.97 (2.00)	**29.70 (1.50)**	**40.53 (2.00)**	**34.80 (1.50)**
	GE2	**45.62 (2.00)**	**43.95 (2.00)**	68.97 (2.50)	35.27 (2.00)	**46.97 (2.00)**	73.43 (3.00)
	GE3	**61.77 (2.00)**	**53.27 (2.00)**	**24.57 (1.50)**	71.53 (2.50)	**49.00 (2.00)**	**28.27 (1.50)**
3	IR1	153.03 (5.00)	146.37 (5.00)	146.97 (5.00)	123.30 (5.00)	140.10 (5.00)	138.60 (5.00)
	IR2	125.63 (5.00)	142.50 (5.00)	123.60 (5.00)	132.73 (5.00)	131.20 (5.00)	117.17 (5.00)
	IR3	127.83 (5.00)	117.63 (5.00)	135.93 (5.00)	150.47 (5.00)	135.20 (5.00)	148.20 (5.00)
	GE1	**33.30 (2.00)**	**18.53 (1.00)**	**41.47 (2.00)**	**17.43 (1.00)**	**20.90 (1.50)**	**45.83 (2.00)**
	GE2	**57.93 (2.00)**	61.07 (2.50)	**45.23 (2.00)**	61.57 (2.50)	49.20 (2.00)	**55.87 (2.00)**
	GE3	**45.27 (2.00)**	56.90 (2.50)	**49.80 (2.00)**	57.50 (2.50)	66.40 (2.50)	**37.33 (2.00)**
5	IR1	138.30 (5.00)	141.97 (5.00)	140.80 (5.00)	123.40 (5.00)	145.07 (5.00)	148.60 (5.00)
	IR2	133.23 (5.00)	148.43 (5.00)	138.93 (5.00)	130.27 (5.00)	117.10 (5.00)	136.73 (5.00)
	IR3	134.97 (5.00)	116.10 (5.00)	126.77 (5.00)	152.83 (5.00)	144.33 (5.00)	121.17 (5.00)
	GE1	**40.40 (2.00)**	**46.50 (2.00)**	**21.17 (1.50)**	**44.00 (2.00)**	**45.00 (2.00)**	39.80 (2.00)
	GE2	**39.00 (2.00)**	**45.00 (2.00)**	53.07 (2.00)	**44.00 (2.00)**	**46.50 (2.00)**	**24.30 (1.50)**
	GE3	**57.10 (2.00)**	**45.00 (2.00)**	62.27 (2.50)	**48.50 (2.00)**	**45.00 (2.00)**	72.40 (2.50)

As Tabelas 7.14 e 7.15 apresentam os resultados para 20 e 60 variáveis na comparação entre a GE e o IRACE utilizando a gramática recursiva para geração de MOPSOs. Da mesma

forma que ocorreu na versão não recursiva, os MOPSOs gerados pela GE se mostram superiores ao obter melhores resultados considerando a distribuição do ranques nas tabelas. Novamente, não houve casos onde o IRACE obteve melhores resultados ou empates com a GE, demonstrando que a Evolução Gramatical proporciona a criação de MOPSOs melhores que o IRACE.

Comparando estes dois algoritmos de geração automática foi possível analisar e constatar, que apesar da GE ter se destacado obtendo melhores resultados com os MOPSOs gerados, ambos os algoritmos têm o potencial de gerar algoritmos eficientes, sendo necessária apenas a escolha de bons componentes e parâmetros para a geração do algoritmo considerado.

7.5.3 Comparação entre MOPSOs gerados automaticamente e SMPSO

Tendo definido o método que obteve os melhores MOPSOs gerados automaticamente um conjunto final de experimentos foi realizado para comparar os melhores MOPSOs gerados com o SMPSO, algoritmo conhecido pelo seu bom desempenho dentre os algoritmos baseados em enxame, assim como o PSO e o MOPSO. As Tabelas 7.16 e 7.17 apresentam os resultados para a comparação entre os MOPSO gerados pela gramática não recursiva, e as Tabelas 7.18 e 7.19 mostram os resultados da comparação dos MOPSOs que utilizaram a gramática recursiva.

Tabela 7.16: Comparação entre melhores MOPSOs e SMPSO para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	GE1	**49.90 (2.00)**	68.08 (3.00)	75.73 (3.00)	63.30 (3.00)	77.72 (3.00)	76.20 (3.50)
	GE2	**60.17 (2.00)**	84.37 (3.00)	48.87 (2.00)	76.00 (3.00)	72.12 (3.00)	51.47 (2.00)
	GE3	87.57 (4.00)	62.85 (3.00)	101.63 (4.00)	81.47 (3.00)	76.67 (3.00)	98.83 (3.50)
	SMPSO	**44.37 (2.00)**	**26.70 (1.00)**	**15.77 (1.00)**	**21.23 (1.00)**	**15.50 (1.00)**	**15.50 (1.00)**
3	GE1	95.03 (3.50)	84.62 (3.50)	56.33 (2.50)	**42.70 (2.00)**	**58.63 (2.50)**	65.47 (2.50)
	GE2	51.87 (2.50)	72.42 (3.00)	72.60 (2.50)	**54.93 (2.00)**	**63.40 (2.50)**	69.57 (3.00)
	GE3	73.77 (3.00)	52.03 (2.00)	97.57 (4.00)	**52.20 (2.00)**	**62.83 (2.50)**	91.47 (3.50)
	SMPSO	**21.33 (1.00)**	**32.93 (1.50)**	**15.50 (1.00)**	92.17 (4.00)	**57.13 (2.50)**	**15.50 (1.00)**
5	GE1	**44.72 (2.00)**	**32.25 (1.50)**	87.07 (3.00)	**45.22 (1.50)**	65.73 (2.50)	79.63 (3.00)
	GE2	65.42 (2.50)	66.30 (2.50)	67.43 (3.00)	**40.20 (1.50)**	**34.87 (1.50)**	70.53 (3.00)
	GE3	82.80 (3.50)	44.22 (2.00)	71.57 (3.00)	75.43 (3.50)	51.47 (2.00)	73.70 (3.00)
	SMPSO	**49.07 (2.00)**	99.23 (4.00)	**15.93 (1.00)**	81.15 (3.50)	89.93 (4.00)	**18.13 (1.00)**

Tabela 7.17: Comparação entre melhores MOPSOs e SMPSO para 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	GE1	78.53 (3.50)	94.03 (3.50)	68.97 (2.50)	74.07 (3.00)	65.63 (2.50)	55.15 (2.00)
	GE2	52.00 (2.00)	55.77 (2.50)	46.93 (2.00)	50.43 (2.00)	68.20 (2.50)	**33.62 (1.50)**
	GE3	95.97 (3.50)	76.70 (3.00)	101.20 (4.00)	102.00 (4.00)	92.10 (4.00)	82.50 (3.50)
	SMPSO	**15.50 (1.00)**	**15.50 (1.00)**	**24.90 (1.50)**	**15.50 (1.00)**	**16.07 (1.00)**	70.73 (3.00)
3	GE1	54.93 (2.50)	84.63 (3.00)	73.63 (3.50)	64.47 (3.00)	89.27 (3.50)	**49.37 (2.00)**
	GE2	65.73 (2.50)	63.43 (3.00)	81.03 (3.50)	71.80 (3.00)	78.07 (3.50)	**57.50 (2.00)**
	GE3	104.37 (4.00)	78.20 (3.00)	**46.27 (1.50)**	87.33 (3.00)	53.63 (2.00)	**37.57 (2.00)**
	SMPSO	**16.97 (1.00)**	**15.73 (1.00)**	**41.07 (1.50)**	**18.40 (1.00)**	**21.03 (1.00)**	97.57 (4.00)
5	GE1	**59.03 (2.50)**	**59.00 (2.50)**	78.67 (3.50)	**59.00 (2.50)**	**60.00 (2.50)**	74.57 (3.00)
	GE2	**59.00 (2.50)**	**59.00 (2.50)**	81.43 (3.50)	**60.97 (2.50)**	**60.00 (2.50)**	63.30 (3.00)
	GE3	**66.97 (2.50)**	**65.00 (2.50)**	54.03 (2.00)	**63.03 (2.50)**	**62.00 (2.50)**	**28.70 (1.00)**
	SMPSO	**57.00 (2.50)**	**59.00 (2.50)**	**27.87 (1.00)**	**59.00 (2.50)**	**60.00 (2.50)**	75.43 (3.00)

Nas Tabelas 7.16 e 7.17 estão os resultados da comparação entre os MOPSOs gerados automaticamente pela GE utilizando a gramática não recursiva para 20 e 60 variáveis. Para 20 variáveis, aparentemente não houve um único algoritmo destacado como o melhor, havendo uma boa distribuição dos ranques. Para 2 objetivos o SMPSO se destacou bastante, obtendo em alguns casos os melhores valores possíveis no ranqueamento com 15.5 e 1.00. Para 3 e 5 objetivos, o SMPSO também se mostrou como um bom algoritmo, porém, nos problemas DTLZ4 e 5 para 3 objetivos, houve vários empates com os MOPSOs da GE. Para 60 variáveis, a situação se altera

um pouco, enquanto que para 2 objetivos o SMPSO aparece em quase todos os problemas como sendo o melhor sem empates, com exceção do DTLZ6, onde o algoritmo GE2 se mostra como o melhor. Para 5 objetivos, na maioria dos problemas houve empate entre todos os 3 MOPSOs gerados e o SMPSO, nos casos onde isso não ocorreu, em um deles o SMPSO foi o melhor, e no outro o GE3 foi o melhor.

Tabela 7.18: Comparação entre melhores MOPSOs recursivos e SMPSO para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	GE1	67.17 (2.50)	77.57 (3.50)	66.47 (3.00)	92.73 (3.50)	99.30 (4.00)	68.10 (2.50)
	GE2	58.27 (2.50)	53.60 (2.00)	79.27 (3.00)	56.37 (2.50)	55.18 (2.50)	57.10 (2.50)
	GE3	101.07 (4.00)	92.33 (3.50)	80.77 (3.00)	71.67 (3.00)	66.15 (2.50)	101.30 (4.00)
	SMPSO	**15.50 (1.00)**	**18.50 (1.00)**	**15.50 (1.00)**	**21.23 (1.00)**	**21.37 (1.00)**	**15.50 (1.00)**
3	GE1	57.90 (3.00)	**50.73 (1.50)**	47.43 (2.00)	**56.25 (2.00)**	65.13 (3.00)	45.50 (2.00)
	GE2	79.93 (3.00)	78.30 (3.50)	97.50 (3.50)	**49.25 (2.00)**	74.10 (3.00)	90.03 (3.50)
	GE3	77.17 (3.00)	77.00 (3.50)	81.57 (3.50)	**49.37 (2.00)**	73.30 (3.00)	90.97 (3.50)
	SMPSO	**27.00 (1.00)**	**35.97 (1.50)**	**15.50 (1.00)**	87.13 (4.00)	**29.47 (1.00)**	**15.50 (1.00)**
5	GE1	**41.05 (2.00)**	**25.50 (1.00)**	**33.67 (1.50)**	**36.53 (1.50)**	**17.23 (1.00)**	50.93 (2.00)
	GE2	88.48 (4.00)	59.05 (2.50)	103.73 (4.00)	85.50 (3.50)	76.20 (3.00)	97.47 (4.00)
	GE3	**63.65 (2.00)**	55.42 (2.50)	76.97 (3.00)	**46.95 (1.50)**	55.13 (2.50)	63.30 (2.50)
	SMPSO	**48.82 (2.00)**	102.03 (4.00)	**27.63 (1.50)**	73.02 (3.50)	93.43 (3.50)	**30.30 (1.50)**

Tabela 7.19: Comparação entre melhores MOPSOs recursivos e SMPSO para 60 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	GE1	50.00 (2.50)	51.55 (2.00)	73.80 (3.00)	81.50 (3.50)	57.50 (2.00)	**41.85 (1.50)**
	GE2	71.03 (2.50)	81.05 (3.50)	98.33 (4.00)	50.67 (2.00)	84.50 (3.50)	79.47 (3.50)
	GE3	105.47 (4.00)	93.90 (3.50)	49.20 (2.00)	94.33 (3.50)	84.50 (3.50)	**29.92 (1.50)**
	SMPSO	**15.50 (1.00)**	**15.50 (1.00)**	**20.67 (1.00)**	**15.50 (1.00)**	**15.50 (1.00)**	90.77 (3.50)
3	GE1	69.87 (3.00)	43.63 (2.00)	**47.67 (1.50)**	50.57 (2.00)	45.70 (2.00)	**55.33 (2.00)**
	GE2	80.60 (3.00)	95.67 (3.50)	83.53 (3.50)	94.40 (3.50)	85.33 (3.50)	**42.17 (2.00)**
	GE3	74.30 (3.00)	85.33 (3.50)	78.67 (3.50)	77.83 (3.50)	91.67 (3.50)	**46.43 (2.00)**
	SMPSO	**17.23 (1.00)**	**17.37 (1.00)**	**32.13 (1.50)**	**19.20 (1.00)**	**19.30 (1.00)**	98.07 (4.00)
5	GE1	**54.35 (2.00)**	**61.52 (2.50)**	**34.27 (1.50)**	**60.00 (2.50)**	**60.50 (2.50)**	**28.13 (1.50)**
	GE2	**52.50 (2.00)**	**59.50 (2.50)**	66.57 (3.00)	**60.00 (2.50)**	**60.50 (2.50)**	**41.57 (1.50)**
	GE3	80.83 (4.00)	**61.48 (2.50)**	104.43 (4.00)	**62.00 (2.50)**	**60.50 (2.50)**	104.50 (4.00)
	SMPSO	**54.32 (2.00)**	**59.50 (2.50)**	**36.73 (1.50)**	**60.00 (2.50)**	**60.50 (2.50)**	67.80 (3.00)

As Tabelas 7.18 e 7.19 apresentam os resultados da comparação entre os MOPSOs gerados pela GE usando a gramática recursiva e o SMPSO. Para 20 e 60 variáveis o comportamento apresentado para 2 e 3 objetivos é semelhante. Para 2 objetivos o SMPSO aparece na maioria dos problemas como o melhor algoritmo. Para 3 objetivos, apesar do SMPSO ainda aparecer mais vezes como o melhor, ocorreram mais empates em alguns problemas. Para 5 objetivos houve um aumento na quantidade de empates, e para 60 variáveis isso ocorre com mais frequência ainda.

Sem dúvida o SMPSO demonstrou possuir um ótimo desempenho nos problemas de *benchmark* considerados, porém, os MOPSOs gerados tanto pela gramática normal quanto a recursiva, apresentaram resultados próximos principalmente nos cenários com 5 objetivos e 60 variáveis. A tendência apresentada, sugere que os MOPSOs gerados automaticamente foram melhorando seu desempenho à medida que a complexidade das instâncias foi aumentando.

7.5.4 Análise da frequência dos componentes e tempo de execução

Analisando todos os MOPSOs gerados durante a fase de treinamento dos algoritmos GE e IRACE, a Tabela 7.20 apresenta a frequência dos componentes e parâmetros escolhidos durante a maioria dos casos durante a criação e otimização dos MOPSOs. Os resultados foram divididos de acordo com o algoritmo gerador utilizado (GE ou IRACE).

Tabela 7.20: Frequência dos componentes e parâmetros utilizados

	GE		IRACE	
Parâmetro/ Componente	**Valor**	**Freq.**	**Valor**	**Freq.**
População	70	56%	70	100%
Repositório	Crowding Distance	56%	Crowding Distance	78%
Att Rep.	Out Loop	67%	In Loop	67%
Líder	Sigma	100%	NW Sum	56%
Velocidade	Speed Constraint	100%	Speed Constraint / Constriction	33%
C1	1.25-2.25	33%	1-2 / 1.5-2.5	33%
C2	1.75-2.75	33%	2-3	44%
Inercia	0.1 / 0.7	22%	0.9	50%
Mutação	Polynomial	67%	Polynomial	78%
Prob Mut.	0.025	44%	0.01	33%

De acordo com os componentes e parâmetros utilizados mais frequentemente na criação dos MOPSOs pela GE, na maioria dos casos, os algoritmos gerados assemelham-se a configuração utilizada pelo SMPSO neste trabalho. Com exceção do método de seleção de líder, em que o SMPSO utilizou *Crowding Distance*, C_1 e C_2 que eram definidos pelo intervalo de 1.5 e 2.5 e a probabilidade de mutação definida como 1%, os demais componentes e parâmetros foram os mesmos. O mesmo comportamento não ocorre com os MOPSOs gerados pelo IRACE, o que talvez tenha influenciado o desempenho destes algoritmos, que se mostraram inferiores ao desempenho dos MOPSOs da GE de acordo com a avaliação realizada.

Tabela 7.21: Comparação do tempo de execução dos algoritmos de projeto

	Objetivos		
Algoritmos	**2**	**3**	**5**
IRACE	9:40h	17:21h	24:14h
GE	13:29h	17:05h	87:19h

Na Tabela 7.21 estão presentes os tempos médios que cada técnica de projeto de algoritmos levou para executar o treinamento. Os experimentos foram executados com um computador Intel Core i7-5930K com 3.5GHz e 40Gb de memória. Em geral, o IRACE executou muito mais rápido que a GE. O IRACE tende a ser mais rápido devido ao *budget* de execução. O *budget* completo somente é usado quando necessário, e cada projeto de MOPSO é executado somente o necessário para constatar diferença estatística para com os demais.

Tendo em vista que o IRACE levou consideravelmente menos tempo para o treinamento, pensou-se na hipótese de se obter melhores resultados para um treinamento mais longo. Pensando nisso, uma nova fase de treino foi executada somente para o IRACE dessa vez considerando o dobro de *budget*, nesse caso 40000, para verificar se com mais tempo, o IRACE consegue obter melhores MOPSOs.

Após a execução do treinamento foram geradas as Tabelas 7.22 e 7.23 onde foram comparados os MOPSOs gerados pelo IRACE para 20000 e 40000 de *budget*, e as Tabelas 7.24 e 7.25, mostram a comparação entre os MOPSOs gerados pelo IRACE com os resultados anteriores da GE.

Os resultados presentes nas Tabelas 7.22 e 7.23 mostram a comparação entre os resultados obtidos pelo IRACE treinado com 20000 (A) e 40000 (B) de *budget*. Com exceção dos

Tabela 7.22: Comparação entre IRACE treinado com 20000 e 40000 de *budget* para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1 A	74.67 (3.00)	127.90 (5.50)	90.60 (3.50)	**49.03 (2.00)**	**110.30 (3.50)**	118.80 (4.50)
	IR2 A	94.00 (3.50)	89.33 (3.00)	**86.10 (3.00)**	108.33 (4.00)	**74.27 (3.50)**	97.43 (4.50)
	IR3 A	112.13 (4.00)	78.93 (3.00)	**83.23 (3.00)**	124.77 (5.00)	**96.63 (3.50)**	102.90 (4.50)
	IR1 B	118.30 (5.00)	**55.80 (2.50)**	**77.53 (3.00)**	105.60 (4.00)	**79.40 (3.50)**	**57.73 (1.50)**
	IR2 B	76.97 (3.00)	77.73 (3.00)	**77.00 (3.00)**	83.53 (3.00)	**107.03 (3.50)**	**56.17 (1.50)**
	IR3 B	**66.93 (2.50)**	113.30 (4.00)	128.53 (5.50)	71.73 (3.00)	**75.37 (3.50)**	109.97 (4.50)
3	IR1 A	92.30 (3.50)	**69.67 (3.50)**	117.60 (4.50)	**83.60 (3.50)**	**35.10 (1.50)**	62.10 (2.50)
	IR2 A	111.07 (4.50)	**107.77 (3.50)**	105.30 (3.50)	**92.90 (3.50)**	115.73 (4.50)	98.53 (3.50)
	IR3 A	97.13 (3.50)	**78.83 (3.50)**	94.53 (3.50)	**99.90 (3.50)**	122.37 (5.00)	123.00 (5.00)
	IR1 B	**69.30 (2.50)**	**99.00 (3.50)**	**73.47 (3.00)**	**72.07 (3.50)**	78.73 (3.00)	76.40 (3.00)
	IR2 B	113.17 (4.50)	**96.13 (3.50)**	**72.50 (3.00)**	**107.63 (3.50)**	120.93 (5.00)	**34.77 (1.50)**
	IR3 B	**60.03 (2.50)**	**91.60 (3.50)**	79.60 (3.50)	**86.90 (3.50)**	70.13 (2.00)	148.20 (5.50)
5	IR1 A	125.33 (5.00)	96.97 (3.50)	**47.80 (2.50)**	107.43 (4.50)	**30.10 (1.00)**	101.90 (4.50)
	IR2 A	**63.77 (2.50)**	117.13 (4.50)	111.07 (4.00)	82.77 (3.50)	74.07 (3.50)	**42.77 (1.50)**
	IR3 A	84.63 (3.00)	98.97 (3.50)	85.73 (3.00)	120.20 (4.50)	111.60 (4.00)	109.27 (4.50)
	IR1 B	99.67 (3.50)	**66.13 (3.00)**	148.63 (5.50)	**65.20 (2.00)**	74.70 (3.50)	99.53 (4.50)
	IR2 B	66.67 (3.00)	85.73 (3.50)	76.93 (3.00)	110.00 (4.50)	144.73 (5.00)	133.10 (4.50)
	IR3 B	102.93 (4.00)	**78.07 (3.00)**	72.83 (3.00)	**57.40 (2.00)**	107.80 (4.00)	**56.43 (1.50)**

Tabela 7.23: Comparação entre IRACE treinado com 20000 e 40000 de *budget* para 40 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1 A	122.43 (4.50)	89.43 (3.00)	**49.03 (1.50)**	66.40 (3.00)	79.57 (2.50)	101.77 (4.50)
	IR2 A	**42.90 (1.50)**	**30.47 (1.50)**	111.13 (4.50)	89.90 (3.50)	**45.63 (2.00)**	121.43 (5.00)
	IR3 A	113.73 (4.50)	65.90 (2.50)	120.53 (4.50)	92.23 (3.50)	94.97 (4.00)	**48.43 (2.00)**
	IR1 B	**52.77 (1.50)**	143.97 (5.50)	**53.43 (1.50)**	**47.20 (1.50)**	129.63 (5.00)	**54.03 (2.00)**
	IR2 B	97.90 (4.50)	78.50 (3.00)	109.67 (4.50)	100.67 (3.50)	61.77 (2.50)	136.47 (5.00)
	IR3 B	113.27 (4.50)	134.73 (5.50)	99.20 (4.50)	146.60 (6.00)	131.43 (5.00)	80.87 (2.50)
3	IR1 A	130.10 (5.00)	95.03 (4.00)	76.10 (3.00)	110.30 (4.00)	113.07 (5.00)	108.03 (4.50)
	IR2 A	76.20 (2.50)	**44.07 (1.50)**	118.10 (5.00)	78.07 (3.50)	74.70 (2.50)	135.43 (5.50)
	IR3 A	**40.60 (2.00)**	96.60 (4.00)	**57.37 (2.00)**	107.07 (4.00)	**48.87 (2.00)**	**63.47 (2.50)**
	IR1 B	75.53 (2.50)	**50.43 (1.50)**	113.80 (4.00)	95.07 (4.00)	106.13 (4.50)	94.50 (3.00)
	IR2 B	115.80 (5.00)	140.53 (5.50)	100.30 (4.00)	122.20 (4.50)	143.93 (5.00)	74.93 (3.00)
	IR3 B	104.77 (4.00)	116.33 (4.50)	77.33 (3.00)	**30.30 (1.00)**	**56.30 (2.00)**	**66.63 (2.50)**
5	IR1 A	110.47 (4.50)	77.77 (3.00)	**78.60 (3.50)**	**59.80 (2.50)**	**42.67 (1.50)**	109.67 (4.00)
	IR2 A	137.47 (5.50)	158.20 (5.50)	**107.17 (3.50)**	86.50 (3.50)	115.50 (4.50)	129.73 (5.50)
	IR3 A	88.93 (3.00)	122.10 (5.00)	**72.83 (3.50)**	82.43 (3.50)	85.20 (3.50)	83.30 (3.00)
	IR1 B	**59.73 (2.50)**	58.60 (2.00)	**93.17 (3.50)**	95.57 (3.50)	66.57 (2.00)	**50.40 (2.50)**
	IR2 B	**55.50 (2.50)**	**24.17 (1.50)**	**93.80 (3.50)**	111.67 (4.00)	106.03 (4.50)	83.50 (3.00)
	IR3 B	90.90 (3.00)	102.17 (4.00)	**97.43 (3.50)**	107.03 (4.00)	127.03 (5.00)	86.40 (3.00)

poucos casos onde praticamente todos os algoritmos empataram, é difícil destacar qual obteve melhor desempenho, pois em ambos os casos há em torno de 1 ou 2 algoritmos que se destacam, seja para a A ou B.

As Tabelas 7.24 e 7.25 apresentam os resultados da comparação entre o IRACE com o treinamento de 40000 de *budget* e os algoritmos obtidos com a GE no treinamento com 20000. Mesmo com mais tempo e recursos, o IRACE não conseguiu gerar MOPSOs que obtivessem resultados mais próximos da GE, indicando que o IRACE está convergindo mais rapidamente, sendo difícil conseguir melhorar os algoritmos a partir de certo ponto.

7.5.5 Considerações finais

No geral, a auto configuração de algoritmos mostrou que tem potencial de gerar bons algoritmos. Através do primeiro conjunto de experimentos foi possível perceber que simples mudanças na gramática provocaram grande diferença no desempenho dos algoritmos. Neste caso, a alteração tornou os projetos de MOPSO mais próximos do algoritmo referência, o SMPSO, havendo a possibilidade serem gerados algoritmos idênticos a ele em configuração.

Tabela 7.24: Comparação entre IRACE treinado com 40000 de *budget* e GE para 20 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	145.93 (5.00)	148.80 (5.00)	137.43 (5.00)	130.23 (5.00)	140.83 (5.00)	121.30 (4.50)
	IR2	133.87 (5.00)	117.60 (5.00)	116.50 (5.00)	127.93 (5.00)	131.97 (5.00)	124.20 (5.00)
	IR3	126.70 (5.00)	140.10 (5.00)	152.57 (5.00)	148.33 (5.00)	133.70 (5.00)	161.00 (5.50)
	GE1	**45.03 (2.00)**	**49.83 (2.00)**	45.20 (2.00)	**32.45 (2.00)**	**34.73 (2.00)**	**39.37 (2.00)**
	GE2	**38.33 (2.00)**	**39.27 (2.00)**	**22.20 (1.50)**	**52.00 (2.00)**	**43.63 (2.00)**	**33.63 (2.00)**
	GE3	**53.13 (2.00)**	**47.40 (2.00)**	69.10 (2.50)	**52.05 (2.00)**	**58.13 (2.00)**	**63.50 (2.00)**
3	IR1	113.50 (5.00)	127.07 (5.00)	142.63 (5.00)	145.13 (5.00)	149.03 (5.00)	137.77 (5.00)
	IR2	145.47 (5.00)	130.83 (5.00)	133.33 (5.00)	123.47 (5.00)	128.60 (5.00)	146.10 (5.00)
	IR3	147.53 (5.00)	148.60 (5.00)	130.53 (5.00)	137.90 (5.00)	128.87 (5.00)	122.63 (5.00)
	GE1	**60.98 (2.00)**	**51.12 (2.00)**	**24.13 (2.00)**	**61.57 (2.00)**	**41.47 (2.00)**	**52.45 (2.00)**
	GE2	**46.38 (2.00)**	**34.40 (2.00)**	**54.90 (2.00)**	**32.67 (2.00)**	**46.13 (2.00)**	**27.40 (2.00)**
	GE3	**29.13 (2.00)**	**50.98 (2.00)**	**57.47 (2.00)**	**42.27 (2.00)**	**48.90 (2.00)**	**56.65 (2.00)**
5	IR1	113.17 (5.00)	149.33 (5.00)	117.30 (4.50)	128.23 (5.00)	142.13 (5.00)	110.23 (4.50)
	IR2	147.33 (5.00)	141.50 (5.00)	158.50 (5.50)	151.30 (5.00)	151.70 (5.50)	154.60 (5.50)
	IR3	146.00 (5.00)	115.67 (5.00)	130.70 (5.00)	126.97 (5.00)	112.67 (4.50)	141.67 (5.00)
	GE1	**37.08 (2.00)**	**47.65 (2.00)**	**54.77 (2.00)**	**41.20 (2.00)**	**27.05 (2.00)**	**56.57 (2.00)**
	GE2	**31.07 (2.00)**	**55.13 (2.00)**	**35.23 (2.00)**	**51.53 (2.00)**	**64.90 (2.00)**	**47.20 (2.00)**
	GE3	**68.35 (2.00)**	**33.72 (2.00)**	**46.50 (2.00)**	**43.77 (2.00)**	**44.55 (2.00)**	**32.73 (2.00)**

Tabela 7.25: Comparação entre IRACE treinado com 40000 de *budget* e GE para 40 variáveis

Obj.	Algorithms	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6
2	IR1	146.57 (5.00)	140.50 (5.00)	99.63 (4.50)	121.10 (5.00)	142.53 (5.00)	128.90 (5.00)
	IR2	137.77 (5.00)	125.80 (5.00)	122.23 (4.50)	127.83 (5.00)	136.47 (5.00)	126.40 (5.00)
	IR3	122.17 (5.00)	140.20 (5.00)	136.63 (4.50)	157.57 (5.00)	127.50 (5.00)	151.20 (5.00)
	GE1	44.87 (2.00)	**53.43 (2.00)**	**45.23 (1.50)**	**35.77 (2.00)**	**33.47 (2.00)**	**36.57 (2.00)**
	GE2	**20.03 (1.50)**	**38.73 (2.00)**	**18.00 (1.50)**	**32.67 (2.00)**	**36.90 (2.00)**	**41.83 (2.00)**
	GE3	71.60 (2.50)	**44.33 (2.00)**	121.27 (4.50)	**68.07 (2.00)**	**66.13 (2.00)**	**58.10 (2.00)**
3	IR1	143.33 (5.00)	127.00 (5.00)	144.73 (5.00)	123.33 (5.00)	124.90 (5.00)	118.50 (5.00)
	IR2	112.27 (4.50)	140.17 (5.00)	126.60 (5.00)	148.17 (5.00)	152.90 (5.00)	147.07 (5.00)
	IR3	150.90 (5.50)	139.33 (5.00)	135.17 (5.00)	135.00 (5.00)	128.70 (5.00)	137.97 (5.00)
	GE1	**30.80 (2.00)**	**31.97 (2.00)**	**46.43 (2.00)**	**43.43 (2.00)**	**48.07 (2.00)**	**43.87 (2.00)**
	GE2	**42.83 (2.00)**	**37.10 (2.00)**	**58.13 (2.00)**	**36.20 (2.00)**	**37.80 (2.00)**	**46.97 (2.00)**
	GE3	**62.87 (2.00)**	**67.43 (2.00)**	**31.93 (2.00)**	**56.87 (2.00)**	**50.63 (2.00)**	**48.63 (2.00)**
5	IR1	160.83 (5.50)	148.17 (5.00)	145.67 (5.00)	123.13 (5.00)	136.50 (5.00)	139.97 (5.00)
	IR2	110.37 (4.50)	124.57 (5.00)	124.37 (5.00)	124.47 (5.00)	160.57 (5.50)	117.30 (5.00)
	IR3	135.30 (5.00)	133.77 (5.00)	136.47 (5.00)	158.90 (5.00)	109.43 (4.50)	149.23 (5.00)
	GE1	**43.50 (2.00)**	**45.50 (2.00)**	**47.00 (2.00)**	**43.98 (2.00)**	**46.00 (2.00)**	47.83 (2.00)
	GE2	**47.95 (2.00)**	**45.50 (2.00)**	**54.07 (2.00)**	**48.53 (2.00)**	**46.00 (2.00)**	67.43 (2.50)
	GE3	**45.05 (2.00)**	**45.50 (2.00)**	**35.43 (2.00)**	**43.98 (2.00)**	**44.50 (2.00)**	**21.23 (1.50)**

Durante o segundo conjunto de experimentos, colocamos dois algoritmos de configuração automática de algoritmos para competir entre si, duas propostas diferentes entre algoritmos evolutivos e avaliação através de testes estatísticos em uma "corrida". Neste caso, o algoritmo evolutivo conseguiu mostrar sua habilidade de melhorar os algoritmos avaliados enquanto que o IRACE eliminava as piores configurações ao longo de diferentes amostras.

No último conjunto de experimentos, onde colocamos à prova os melhores MOPSOs configurados automaticamente e o algoritmo referência SMPSO, foi possível perceber que mesmo não se destacando como o melhor algoritmo na maioria dos casos, os MOPSOs conseguiam sim competir com o SMPSO, obtendo resultados competitivos e inclusive alguns casos com melhores resultados, demonstram o potencial que algoritmos configurados automaticamente possuem.

Em experimentos adicionais, realizados com o objetivo de verificar se o IRACE conseguiria obter melhores resultados caso tivesse mais tempo e recursos, observou-se que o desempenho médio dos algoritmos gerados pelo IRACE não apresentou melhores significativas mesmo considerando o dobro da *budget* no treinamento, concluindo que a GE, mesmo tomando mais tempo para executar o treinamento, consegue gerar MOPSOs mais eficientes que o IRACE.

Capítulo 8

Conclusão

Dentro da área de otimização cada vez mais algoritmos são criados para a resolução dos mais simples aos mais complexos problemas presentes diariamente na nossa vida. Para que qualquer um destes algoritmos tenha um desempenho bom o suficiente para ser aplicado, ele deve passar por diversos testes de escolha e refinamento para seus parâmetros. Devido a este ajuste necessário dos parâmetros, realizar essa tarefa manualmente pode tomar muito tempo e recursos, razão pela qual foi considerado cada vez mais a configuração automática para estes algoritmos.

O estudo presente nessa dissertação toma como base outros trabalhos da área que utilizam diferentes técnicas para a parametrização e criação de algoritmos de forma automática. Trabalhos como o de Contreras-Bolton e Parada [13] que propõe a utilização de programação genética para a criação e otimização de algoritmos de *clustering* e também para o problema do Caixeiro Viajante; e Miranda e Prudêncio [33] que propuseram um *framework* para a otimização do algoritmo PSO, escolhendo seus componentes e parâmetros através da Evolução Gramatical, serviram como base para essa dissertação que propõe a aplicação da Evolução Gramatical para realizar a otimização do algoritmo MOPSO.

O algoritmo MOPSO é uma extensão do PSO que foi criada para considerar problemas com múltiplos objetivos, sendo necessário a adição de componentes conhecidos como operadores de repositório e seleção de líder, usados para armazenar e selecionar a melhor solução dentre um conjunto respectivamente. Partes de sua estrutura, componentes e parâmetros foram selecionados para constituírem uma gramática que através de suas regras de produção permitiriam a criação de diferentes MOPSOs com diferentes componentes e parâmetros. A otimização através da Evolução Gramatical utiliza a gramática para gerar sua população de programas para então executá-los em um problema ou conjunto de problemas definidos. Os problemas considerados foram o conjunto DTLZ, onde cada problema possui suas próprias características.

Assim, foi desenvolvido um *framework* que realiza a configuração automática de algoritmos, podendo realizar a otimização através da Evolução Gramatical ou o IRACE. Experimentos foram realizados para avaliar diversos pontos do *framework* apresentado. Foram comparadas diferentes gramáticas criadas durante o estudo, buscando conhecimento sobre quais componentes e parâmetros causariam maior impacto no projeto de algoritmos. Também foi realizada a comparação entre os métodos de auto-configuração, para eleger aquele que melhor consegue gerar algoritmos genéricos o suficiente para obter bons resultados dentro dos problemas considerados, e manter esse desempenho quando submetidos a problemas desconhecidos. Por fim, os melhores MOPSOs gerados automaticamente foram selecionados e comparados com o algoritmo SMPSO. Os MOPSOs analisados atingiram resultados competitivos em comparação ao SMPSO, demonstrando o potencial que técnicas de auto-configuração possuem.

De acordo com os experimentos realizados, foi possível analisar o impacto que cada técnica para geração dos MOPSOs teve ao comparar os resultados dos melhores MOPSOs gerados com os resultados do SMPSO. Apesar do SMPSO ter apresentado vantagem na maioria dos cenários testados, os algoritmos gerados pela GE demonstraram grande potencial, principalmente em instâncias de problemas mais complexas, havendo casos inclusive onde estes algoritmos superam o desempenho do SMPSO, deixando claro que a configuração feita de forma automática possui resultados competitivos com outros algoritmos. Não se pode afirmar que ele seja melhor ou pior, pois nestes experimentos somente foram analisados os resultados para com o SMPSO, sendo necessária a execução de mais experimentos com outros algoritmos para verificar se o bom desempenho se mantém.

A abordagem ainda apresenta diversas limitações, porém, ainda existem muitas oportunidades de estudo sobre o projeto automático de algoritmos. Dentre elas podemos destacar:

- Realização de mais experimentos, comparando os resultados do GEMOPSO com outros algoritmos bem conceituados na literatura;
- A adição de diferentes componentes e parâmetros para aumentar o espaço de possibilidades de algoritmos a serem gerados a partir da gramática;
- A refatoração do GEMOPSO para permitir seu uso por meio de uma arquitetura distribuída, com o objetivo de diminuir seu tempo de execução elevado;
- Testar o desempenho do GEMOPSO na geração de outros algoritmos, sendo apenas necessária a modificação da gramática empregada, e do módulo que realiza o mapeamento da solução para o algoritmo executável.

O estudo sobre configuração automática de algoritmos ainda possui muitas características a serem exploradas, e vários hipóteses a serem testadas, onde cada nova descoberta é um passo a mais em direção a um futuro mais inteligente.

Referências Bibliográficas

[1] J. E. Alvarez-Benitez, R. M. Everson, and J. E. Fieldsend. A MOPSO Algorithm based Exclusively on Pareto Dominance Concepts. In *Proceedings of the 2005 Congress on Evolutionary Multi-Criterion Optimization (EMO'05)*, pages 459–473. Springer, 2005.

[2] P. Balaprakash, M. Birattari, and T. Stützle. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. In *Proceedings of the 2007 Congress on International Workshop on Hybrid Metaheuristics (HM'07)*, pages 108–122. Springer, 2007.

[3] M. Birattari and J. Kacprzyk. *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197. Springer, 2009.

[4] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and Iterated F-Race: An Overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[5] A. Britto and A. Pozo. Using Archiving Methods to Control Convergence and Diversity for Many-objective Problems in Particle Swarm Optimization. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC'12)*, pages 1–8. IEEE, 2012.

[6] E. K. Burke, M. R. Hyde, and G. Kendall. Grammatical Evolution of Local Search Heuristics. *Evolutionary Computation, IEEE Transactions on*, 16(3):406–417, 2012.

[7] J. Byrne, M. Fenton, E. Hemberg, J. McDermott, and M. O Neill. Optimising Complex Pylon Structures with Grammatical Evolution. *Information Sciences*, 316:582–597, 2015.

[8] O. R. Castro and A. Pozo. A MOPSO based on Hyper-Heuristic to Optimize Many-objective Problems. In *Proceedings of the 2014 IEEE Symposium on Swarm Intelligence (SIS'14)*, pages 1–8. IEEE, 2014.

[9] O. R. Castro Jr and A. Pozo. Using Hyper-Heuristic to Select Leader and Archiving Methods for Many-Objective Problems. In *Proceedings of the 2015 Congress on Evolutionary Multi-Criterion Optimization (EMO'17)*, pages 109–123. Springer, 2015.

[10] R. Cerri, R. C. Barros, A. C. de Carvalho, A. Freitas, et al. A Grammatical Evolution Algorithm for Generation of Hierarchical Multi-label Classification Rules. In *Proceedings of the 2013 Congress on Evolutionary Computation (CEC'13)*, pages 454–461. IEEE, 2013.

[11] M. Clerc and J. Kennedy. The Particle Swarm-explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[12] C. A. C. Coello and M. S. Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, volume 2, pages 1051–1056. IEEE, 2002.

[13] C. Contreras-Bolton and V. Parada. Automatic Design of Algorithms for Optimization Problems. In *Proceedings of the 2nd Latin-American Congress on Computational Intelligence and 12ˆo Congresso Brasileiro de Inteligência Computacional (LACCI'15, CBIC'15)*, pages 3–8, 2015.

[14] V. D. da Fontoura, R. H. R. de Lima, A. T. R. Pozo, and R. S. Hermida. Algoritmos Multiobjetivos aplicados ao Problema de Predição de Estruturas Proteínas. In C. J. A. Bastos Filho, A. R. Pozo, and H. S. Lopes, editors, *Anais do 12 Congresso Brasileiro de Inteligência Computacional*, pages 1–6, Curitiba, PR, 2015. ABRICOM.

[15] V. D. da Fontoura, R. H. R. de Lima, A. T. R. Pozo, and R. S. Hermida. Multi-objective approach to the Protein Structure Prediction Problem. In *Evolutionary Multi-objective System Design*, 2015. Em produção.

[16] R. H. R. de Lima and A. T. R. Pozo. A Study on Auto-configuration of Multi-objective Particle Swarm Optimization Algorithm. In *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC'17)*. IEEE, 2017. Aceito para publicação.

[17] K. Deb. Multi-objective Optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[19] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. *Scalable Test Problems for Evolutionary Multiobjective Optimization*. Springer, 2005.

[20] M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[21] R. C. Eberhart, J. Kennedy, et al. A New Optimizer Using Particle Swarm Theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS'95)*, volume 1, pages 39–43. New York, NY, 1995.

[22] R. C. Eberhart and Y. Shi. Particle Swarm Optimization: Developments, Applications and Resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC'01)*, volume 1, pages 81–86. IEEE, 2001.

[23] Z. Garner. An Introduction to Genetic Programming. 6 2000.

[24] N. Hallam, P. Blanchfield, and G. Kendall. Handling Diversity in Evolutionary Multiobjective Optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC'05)*, volume 3, pages 2233–2240. IEEE, 2005.

[25] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* U Michigan Press, 1975.

[26] J. Kennedy. Particle Swarm Optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[27] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proceedings of 1095 IEEE International Conference on Neural Network*, pages 1942–1948. sn, 1995.

[28] W. H. Kruskal and W. A. Wallis. Use of Ranks in One-criterion Variance Analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[29] M. Laumanns and R. Zenklusen. Stochastic Convergence of Random Search Methods to Fixed Size Pareto Front Approximations. *European Journal of Operational Research*, 213(2):414–421, 2011.

[30] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The Irace Package, Iterated Race for Automatic Algorithm Configuration. Technical report, Citeseer, 2011.

[31] N. Lourenço, F. Pereira, and E. Costa. Evolving Evolutionary Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'12)*, pages 51–58. ACM, 2012.

[32] S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2014.

[33] P. B. Miranda and R. B. Prudêncio. GEFPSO: A Framework for PSO Optimization based on Grammatical Evolution. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 1087–1094. ACM, 2015.

[34] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT press, 1998.

[35] S. Mostaghim and J. Teich. Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS'03)*, pages 26–33. IEEE, 2003.

[36] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. Coello, F. Luna, and E. Alba. SMPSO: A New PSO-based Metaheuristic for Multi-objective Optimization. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-criteria Decision-Making (MCDM'09)*, pages 66–73. IEEE, 2009.

[37] F. Neri and V. Tirronen. Recent Advances in Differential Evolution: A Survey and Experimental Analysis. *Artificial Intelligence Review*, 33(1-2):61–106, 2010.

[38] M. O'Neill and C. Ryan. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[39] N. Padhye, J. Branke, and S. Mostaghim. Empirical Comparison of MOPSO Methods-guide Selection and Diversity Preservation. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC'09)*, pages 2516–2523. IEEE, 2009.

[40] R. Poli, J. Kennedy, and T. Blackwell. Particle Swarm Optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[41] R. Poli and J. Koza. Genetic Programming. In *Search Methodologies*, pages 143–185. Springer, 2014.

[42] R. Poli, W. B. Langdon, and O. Holland. *Extending Particle Swarm Optimisation via Genetic Programming*. Springer, 2005.

[43] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. *A Field Guide to Genetic Programming*. Lulu. com, 2008.

[44] F. Rothlauf. *Design of Modern Heuristics: Principles and Application*. Springer Science & Business Media, 2011.

[45] C. Ryan, J. Collins, and M. O. Neill. Grammatical Evolution: Evolving Programs for an Arbitrary Language. In *Genetic Programming*, pages 83–96. Springer, 1998.

[46] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems. *Evolutionary Computation, IEEE Transactions on*, 17(6):840–861, 2013.

[47] Y. Shi and R. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 69–73. IEEE, 1998.

[48] R. C. Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. 2013, 2014.

[49] P. J. Van Laarhoven and E. H. Aarts. *Simulated Annealing*. Springer, 1987.

[50] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary computation*, 8(2):173–195, 2000.

[51] E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In *Proceedings of the 1998 Congress on Parallel Problem Solving From Nature (PPSN'98)*, pages 292–301. Springer, 1998.