UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO, ADMINISTRAÇÃO E ECONOMIA - DEPRO

ANÁLISE DE DESEMPENHO DE ALGORITMO GENÉTICO EM DIFERENTES LINGUAGENS E AMBIENTES DE PROGRAMAÇÃO

GUILHERME HENRIQUE GUERRA GOMES

Ouro Preto – MG

2015

GUILHERME HENRIQUE GUERRA GOMES

ghgomes@live.com

ANÁLISE DE DESEMPENHO DE ALGORITMO GENÉTICO EM DIFERENTES LINGUAGENS E AMBIENTES DE PROGRAMAÇÃO

Monografia submetida à apreciação da banca examinadora de graduação em Engenharia de Produção da Universidade Federal de Ouro Preto, como parte dos requisitos necessários para a obtenção de grau de graduado em Engenharia de Produção.

Orientador: Profº André Luís Silva

Ouro Preto – MG

2015

Folha de Rosto (verso) – Ficha catalográfica

Deverá ser elaborada pelo profissional bibliotecário de sua Unidade ou da Biblioteca Ce ntral, objetivando a padronização das entradas de autor, orientador e definição dos cabeçalhos de assunto à partir de índices de assuntos reconhecidos internacionalmente.

0xxx

GOMES, Guilherme H. G. Título da monografia. 20XX YY páginas.

Orientador: Profº André Luís Silva.

Monografia (Graduação) – Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Produção, Administração e Economia.

1. Palavra chave. 2. Palavra chave. 3. Palavra chave. 4. Palavra chave. 5. Palavra chave.

I. Universidade Federal de Ouro Preto. Escola de Minas. Departamento de Engenharia de Produção, Administração e Economia. II. Título.

CDU: xxx.x

GUILHERME HENRIQUE GUERRA GOMES

ANÁLISE DE DESEMPENHO DE ALGORITMO GENÉTICO EM DIFERENTES LINGUAGENS E AMBIENTES DE PROGRAMAÇÃO

Monografia julgada e aprovada em 03 de Julho de 2015 como requisito parcial para obtenção do grau de Engenheiro de Produção no curso de Engenharia de Produção da Universidade Federal de Ouro Preto.

BANCA EXAMINADORA

_____

Profº. André Luís Silva
Universidade Federal de Ouro Preto
Orientador

_____

Profº. Magno Silvério Campos
Universidade Federal de Ouro Preto
Examinador

_____

Profº. Helton Cristiano Gomes
Universidade Federal de Ouro Preto
Examinador

_____

Profº. Luciana Sant'Ana Marques
Universidade Federal de Ouro Preto
Examinadora

Dedico este trabalho a todos que me acompanharam durante a minha formação. Colegas de sala, professores e amigos.

AGRADECIMENTOS

A jornada é mais importante que o destino.

Autor Desconhecido

RESUMO

GOMES, Guilherme H. G. **Análise de Desempenho de Algoritmos Genéticos em Diferentes Linguagens e Ambientes de Programação.** 2015. Monografia (Graduação em Engenharia de Produção). Universidade Federal de Ouro Preto.

Este trabalho aborda a heurística algoritmo genético e trata de analisar o desempenho de um algoritmo genético básico em diferentes linguagens de programação e também em diferentes ambientes de programação. Para isto é apresentada uma introdução ao estudo, apresentando a formulação do problema e seus objetivos, uma fundamentação teórica a fim de embasar cientificamente o trabalho, o planejamento pré-experimental, com a metodologia a ser utilizada, os resultados e análises obtidos pelo experimento e, por fim, as conclusões e recomendações finais.

Palavras-chaves: Algoritmo Genético, Otimização, Análise de desempenho, Linguagem de Programação, Ambiente de Programação.

ABSTRACT

GOMES, Guilherme H. G. **Performance Analysis on Genetic Algorithms Using Different Languages and Development Environments.** 2015. Course Work Conclusion (Graduate in Production Engineering). Federal University of Ouro Preto.

This work presents the heuristic genetic algorithm and aims to analyze the performance of a basic genetic algorithm in different programming languages and also in different development environments. Hence is presented an introduction to this study, detailing the problem formulation and its objectives, a theorical study, in order to give this work a scientific approach, the pre-experimental planning, with the methodology to be used, the results and analysis obtained by the experiment and, at last, the conclusions obtained and final recommendations.

Key-words: Genetic Algorithm, Optimization, Performance Analysis, Programming Language, Integrated Development Environment.

LISTA DE FIGURAS

Figura 1 - *Forma geral de uma função em C++* 25

Figura 2 - *Algoritmo de um AG básico* 29

Figura 3 - *Características da Função De Jong* 34

Figura 4 - *Características da Função Easom* 34

Figura 5 - *Características da Função Rosenbrock* 35

Figura 6 - *Características da Função Goldstein-Price* 35

Figura 7 - *Características da Função Ackley* 36

LISTA DE TABELAS

Tabela 1 - *Ambientes de programação e citações na literatura* 21

Tabela 2 - *Alguns dos principais métodos exatos* 27

Tabela 3 - *Alguns dos principais métodos heurísticos* 28

Tabela 4 - *Parâmetros de execução do AG* 33

Tabela 5 - *Teste ANOVA: p-values.* *38*

Tabela 6 - *Diferenças observadas através do Teste de Tukey.* 38

LISTA DE ABREVIATURAS E SIGLAS

AG Algoritmo Genético

LP Linguagem de Programação

OOP Object-Oriented Programming

JVM Java Virtual Machine

IDE Integrated Development Environment

RCP Rich Client Plataform

WTP Web Tools Plataform

HTML Hypertext Markup Language

PHP Hypertext Preprocessor

GHz Gigahertz

RAM Random Acess Memory

ANOVA Analysis of Variance

LISTA DE ANEXOS

Anexo I - *Resultados: Função DeJong* 41

Anexo II - *Resultados: Função Rosenbrock* 41

Anexo III - *Resultados: Função Goldstein-Price* 42

Anexo IV - *Resultados: Função Ackley* 42

Anexo V - *Resultados: Função Easom* 43

Anexo VI - *Valores mínimos, médios e máximos das métricas utilizadas* 43

Anexo VII - *Teste de Fisher: Função Goldstein-Price vs. Tempo de execução* 44

Anexo VIII - *Teste de Fisher: Função Ackley vs. Valor mínimo encontrado* 44

Anexo IX - *Teste de Fisher: Função Ackley vs. Tempo de execução* 45

Anexo X - *Teste de Fisher: Função Easom vs. Valor mínimo encontrado* 45

Anexo XI - *Teste de Mood – Métrica Geração: Função DeJong* 45

Anexo XII - *Teste de Mood – Métrica Geração: Função Ackley* 46

SUMÁRIO

1. INTRODUÇÃO AO ESTUDO 15

1.1 Formulação do Problema 16

1.2 Objetivo 19

1.3 Justificativa do Trabalho 19

1.4 Estrutura do Trabalho 22

2. FUNDAMENTAÇÃO TEÓRICA 23

2.1 Definições Iniciais 23

2.2 Ferramentas de Programação 29

2.3 Comparação entre Linguagens, Ferramentas e Algoritmos 31

3. PLANEJAMENTO EXPERIMENTAL 32

3.1 Planejamento Pré-Experimental 32

4. RESULTADOS E ANÁLISES 37

5. CONCLUSÕES E RECOMENDAÇÕES 39

5.1 Conclusões 39

5.2 Recomendações 40

6. ANEXOS 41

7. **REFERÊNCIAS BIBLIOGRÁFICAS 47**

1. INTRODUÇÃO AO ESTUDO

Problemas de otimização fazem parte do dia a dia das ciências e das engenharias. Os problemas de otimização de sistemas de produção, roteamento de veículos, otimizações numéricas e design de estruturas são apenas alguns dos exemplos. Isso mostra que muitos pesquisadores precisam de um algoritmo de otimização para a solução de problemas que seja robusto nas mais diversas áreas de pesquisa e atuação. Os algoritmos genéticos são uma classe desses.

Os algoritmos genéticos (AGs) são um ramo dos algoritmos evolucionários e como tal, são utilizados como uma ferramenta para se resolver problemas de otimização baseando-se em uma metáfora do processo biológico de evolução natural. Segundo Linden (2008) eles são uma extensão dos algoritmos evolucionários pois utilizam o conceito de simulação da evolução das espécies através de operadores que realizam processos semelhantes à seleção, mutação e reprodução. São esses operadores que, aliados à uma boa modelagem do problema em questão, conseguem obter soluções as quais métodos tradicionais de otimização não são capazes de tratar, como mostra Price (2005) no prefácio de sua obra.

Os algoritmos, tal como os algoritmos genéticos, são implementados em linguagem de programação (LP). As LPs são, de acordo com Brookshear (2005), uma forma de representar uma série ordenada de passos de forma apropriada para que seja transmitido a alguma máquina, ou para que seja lido por outros seres humanos. Existem atualmente centenas de linguagens de programação, que foram surgindo ao longo do avanço da computação e da inteligência computacional. A empresa de análise da qualidade de softwares Tiobe mantém o chamado *TIOBE Programming Community Index*, uma lista online com 235 linguagens de programação, indicando a popularidade e crescimento no uso de cada uma delas. (TIOBE, 2014)

Ao se pesquisar comparações e análises de desempenho entre diferentes LPs destinadas a heurísticas, percebe-se que estes assuntos são pouco narrados na literatura. O mesmo pode ser dito a respeito das análises feitas entre linguagens de programação implementando algum algoritmo genético. Prechelt (2010) é um exemplo da análise de desempenho de linguagens que mostra para a solução de um problema definido por ele, que a LP escolhida para implementar um algoritmo pode afetar o desempenho do mesmo.

Neste mesmo raciocínio, é possível observar o emprego de diferentes linguagens de programação na literatura que tratam da aplicação de algoritmos genéticos em problemas de otimização. Alguns desses exemplos são: Lee (2005) utilizando Python, Linden (2008) utilizando Java, Ponnambalam (2000) utilizando C++, Pizzuti (2012) utilizando MATLAB, Golfeto (2007) utilizando FORTRAN 90/95 e Constantino (2003) utilizando Pascal.

1.1 Formulação do Problema

Os computadores atuais possuem uma vasta genealogia. Da China antiga até o ano de 1970, dispositivos como o ábaco, as máquinas de Pascal, Leibniz e a máquina eletromecânica de George Stibitz eram usados para que se pudesse, mesmo que de forma rudimentar, armazenar dados e realizar algumas operações básicas. A partir da década de 70, a história das máquinas de computação passou a ser fortemente atrelada ao avanço tecnológico. De lá para cá, os primeiros computadores de mesa, que exigiam muito conhecimento teórico e possuíam pouca capacidade de processamento, se transformaram em máquinas robustas e acessíveis, capazes de realizar milhares de operações por segundo e executar uma infinidade de aplicativos. Para Pressman (2009) a capacidade de processamento e armazenagem do hardware atuais representam um grande potencial para a computação.

Atualmente a computação é utilizada para os mais diversos fins. Um deles é a inteligência computacional, definida por Belfiore (2013) como um campo da inteligência artificial que estuda algoritmos inspirados na natureza ou bio-inspirados, como redes neurais, lógica nebulosa e computação evolucionária. Engelbrecht (2007) não obstante desta definição, afirma que a inteligência computacional é o estudo de mecanismos capazes de se adaptar a fim de permitir ou facilitar um comportamento inteligente em ambientes complexos e em contínua mudança.

O que permitiu não só o desenvolvimento da inteligência computacional, mas de toda computação foi o uso da programação de computadores. A programação é basicamente o processo que torna a formulação de um problema computacional em um programa executável, ou como Van-Roy (2004) declara, é dizer como um computador deve fazer o seu trabalho.

Um método de programação, por sua vez, é a utilização de uma linguagem de programação. Esta é uma forma de representar uma série ordenada de passos de forma apropriada, para que seja transmitida a alguma máquina, ou para que seja lida por outros seres humanos. Isto significa que se torna necessário transformar esses passos em um conjunto de instruções fácil de compreender tanto para a máquina quanto para o homem. (BROOKSHEAR, 2005)

Para se escrever utilizando uma linguagem de programação, de forma que um computador possa interpretar o que está sendo escrito, deve-se utilizar um ambiente de programação. Ele é basicamente um programa de computador que reúne algumas características e ferramentas de apoio para a criação de softwares, tendo como objetivo simplificar o desenvolvimento e execução destes.

A otimização, como caracterizada por Fletcher (2000) é uma mistura de métodos clássicos e heurísticas, de teoria e experimento. Ela pode ser estudada como um ramo da matemática pura, porém possui aplicações em quase todos os ramos das ciências e tecnologias. É definida como a ciência do cálculo das "melhores" soluções de um dado problema matemático. A otimização começou a se desenvolver de fato a partir das décadas de 40 e 50 com a introdução de um novo ramo chamado de programação linear (FLETCHER, 2000). Desde então ela vem se modificando para ser capaz de resolver problemas antes considerados intratáveis.

As heurísticas são uma das formas das quais a otimização se utilizou para evoluir. Alguns tipos de problemas são considerados tão complicados que pode ser que não seja possível encontrar uma solução ótima. Em tais situações, ainda é importante encontrar uma solução viável que seja pelo menos razoavelmente próxima da solução ótima. Os métodos heurísticos são utilizados para tal finalidade. Eles provavelmente encontrarão uma excelente solução viável, apesar de que não serão, necessariamente, uma solução ótima para o problema em questão. (HILLIER, 2010)

Os algoritmos evolucionários são considerados heurísticas baseadas no processo de evolução natural. Como algoritmos genéticos são parte dos algoritmos evolucionários, o mesmo então pode ser dito a eles.

Os AGs e suas variações são hoje muito utilizados em diversas aplicações, como exemplificados em Ferreira (2013) para o sequenciamento de partidas em aeroportos,

Galvão (2014) para o desenho de mapas de rede de transporte público, Souza (2012) para otimização de problemas de roteirização, Oliveira (2014) para otimização de projetos de iluminação pública, Arora (2014) para o design de um sistema produtivo, Fiorentino (2014) para controle da dengue, Zhang (2014) para a otimização de um modelo de bateria íon-lítio, entre outros.

As comparações e análises de desempenho entre diferentes LPs destinadas a heurísticas são pouco narradas na literatura. O mesmo pode ser dito a respeito das análises feitas entre LPs implementando os algoritmos genéticos.

Satachi (2013), Lim (2013) e Ishibuchi (2006) são pesquisadores que seguiram este rumo de trabalho, apresentando uma comparação do desempenho de linguagens ou de algoritmos evolucionários para a solução de um problema de otimização.

Com isso, o que se percebe é que mesmo com o grande número de linguagens de programação, não há muito debate dentro da comunidade científica a respeito de qual linguagem de programação se utilizar para implementar um determinado tipo de heurística. Quando este assunto é debatido, porém, os resultados são inconclusivos, como mostra Prechelt (2000), tendo em vista que cada tipo de linguagem possui pontos fortes, fracos e várias peculiaridades.

Dessa forma, se faz necessário haver debates sobre a eficiência das linguagens de programação quando usadas para implementar heurísticas, sendo aqui analisada a heurística AG.

Sendo assim, a pergunta chave desta monografia pode ser assim formulada:

Há diferença de desempenho entre diferentes linguagens de programação em diferentes ambientes de programação para solucionar problemas iguais de otimização com a heurística algoritmo genético?

1.2 Objetivo

1.2.1 Objetivo geral

Este trabalho busca analisar o desempenho das linguagens de programação C++ e Java na implementação de um algoritmo genético para problemas de otimização linear mono-objetivo em dois ambientes de programação. No caso do C++ serão utilizados os softwares Dev-C++ e Code::Blocks. Para o Java os softwares serão o Eclipse e Netbeans.

1.2.2 Objetivos específicos

Com o intuito de compreender melhor o assunto de modo a alcançar o objetivo geral, delimitou-se metas mais específicas. São elas:

- Realizar um levantamento na literatura sobre os algoritmos genéticos de modo a compreender como implementá-lo para a resolução de problemas;
- Levantar informações sobre o comportamento das linguagens de programação escolhidas frente a implementação da heurística Algoritmo Genético;
- Selecionar problemas de otimização a serem solucionados com o AG;
- Modelar um AG para solucionar os problemas escolhidos;
- Programar e executar o AG tanto em C++ como em Java em seus respectivos ambientes de programação;
- Analisar os resultados e oferecer uma conclusão respondendo à pergunta chave deste trabalho.

1.3 Justificativa do Trabalho

As poucas referencias bibliográficas que debatem o desempenho de diferentes linguagens de programação utilizada na implementação de heurísticas abre precedente para se arquitetar mais e melhores investigações, sendo esta constatação uma das justificativas deste trabalho.

Prechelt (2000) foi um dos pesquisadores que se dedicaram a este estudo. Porém, este não analisou a emprego das linguagens para a implementação de heurísticas. Ou seja, há de haver estudos, como aqui proposto, para complementar a literatura sobre o assunto.

A heurística AG foi escolhida para a realização deste trabalho devido a alguns pontos, sendo eles: trata-se de uma heurística fortemente influenciada pelo método de implementação interno da geração de números aleatórios das linguagens de programação, também influenciada pela forma como as rotinas de repetição das linguagens são construídas, e por fim o quanto esta heurística é gerida/moldada pelas estruturas de dados e suas manipulações.

Outro fator muito importante para a escolha da heurística deve-se ao grande escopo de sua aplicação em problemas reais de otimização. Além disso, o Algoritmo Genético é um algoritmo simples e facilmente implementável em diversas linguagens de programação.

No caso da escolha das linguagens de programação para a implementação do algoritmo, tanto o Java quanto o C++ foram escolhidos devido à grande quantidade de trabalhos na literatura onde ambos foram utilizados, além da grande popularidade das duas linguagens no ambiente universitário, empresarial, dentre outros.

Além disso, por serem linguagens padronizadas e independentes de hardware, seus aplicativos podem ser executados em um amplo espectro de sistemas de computador. (DEITEL, 2006)

Para cada uma das linguagens escolhidas serão utilizados dois ambientes de programação distintos. No caso do C++ serão utilizados o dev-C++ e o Code::Blocks. Já para o Java, serão utilizados o Eclipse e o NetBeans. A escolha desses ambientes se deu principalmente pela familiaridade do autor com estes e também devido ao grande número de citações na literatura. A tabela 1 exemplifica alguns destes:

		Nome (ano)	Título
C++	Dev-C++	*Araújo (2008)*	Detecção de Falhas em Sistemas Automatizados com o OpenGL
		Chen (2006)	Extended Method of generic container based on STL
		Fairbairn (2008)	O Manual Dev C++ E Wx Dev C++
		Satav (2011)	A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development
		Verma (2013)	Strain Measurement Using Image Processing
	Code::Blocks	*Delman (2009)*	Development of a System for Teaching C/C++ Using Robots and Open Source Software in a CS1 Course
		Junior (2012)	Problema 8-Puzzle: Análise da solução usando Backtracking e Algoritmos Genéticos
		Oliveira (2013)	Desenvolvimento de uma Biblioteca de Realidade Aumentada Orientada a Objetos baseada no ARToolkit
		Satav (2012)	A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development
Java	Eclipse	*Budinsky (2004)*	Eclipse modeling framework: a developer's guide
		Chen (2005)	Experiences with Eclipse IDE in programming courses
		Geer (2005)	Eclipse becomes the dominant Java IDE
		Satav (2011)	A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development
	NetBeans	*Böck (2009)*	The Definitive Guide to NetBeans Platform
		Boudreau (2002)	NetBeans: The Definitive Guide
		Deitel (2009)	Java: How to Program
		Linden (2008)	Algoritmos Genéticos: Uma Importante Ferramenta da Inteligência Computacional
		Satav (2011)	A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development.

Tabela 1 - *Ambientes de programação e citações na literatura. Fonte: O Autor*

É importante destacar que os problemas de otimização utilizados nas implementações seguiram um critério para serem selecionados. Como neste trabalho não se interessa saber qual a solução de um determinado problema, e sim saber o desempenho do algoritmo em linguagens de programação, serão utilizadas funções de teste, que são funções recorrentemente utilizadas para avaliar a qualidade dos processos de otimização, tendo em vista que são bastante complexas e exigem bastante dos processos para que se obtenha um resultado satisfatório. Para o nosso estudo optou-se por trabalhar com as cinco funções mono-objetivo apresentadas a seguir, sugeridas por Molga (2005) em sua lista de funções de teste:

a. Função De Jong – A chamada primeira função de De Jong é uma das mais simples funções de teste, sendo contínua, convexa e unimodal.
b. Função Easom – É uma função de teste unimodal, onde o mínimo global está em uma pequena área do espaço de busca.
c. Função Rosenbrock – Representa um problema clássico de otimização, onde achar seu vale é trivial, mas encontrar o ótimo global é difícil.

d. Função Goldstein-Price – É uma função de teste de otimização global com apenas duas variáveis.
e. Função Ackley's– É uma função de teste multimodal amplamente usada.

1.4 Estrutura do Trabalho

O trabalho está dividido em cinco capítulos, onde no primeiro capítulo é apresentada a formulação do problema, a justificativa para a realização do trabalho e seus objetivos geral e específicos.

O segundo capítulo trata da fundamentação teórica dos conceitos e teorias que dizem respeito ao assunto aqui tratado. São abordados também as ferramentas de programação a serem utilizadas e finalmente, são mostrados exemplos da literatura que comparam as linguagens, as ferramentas e os algoritmos entre si.

O terceiro capítulo apresenta o planejamento experimental da análise foco do trabalho. O procedimento metodológico é explicado, além de detalhar o planejamento pré-experimento, que detalha os fatores, níveis, variáveis de resposta, materiais e métodos a serem empregados na análise.

O capítulo quatro trata dos resultados do experimento e da análise feita correlacionando os valores encontrados.

Finalmente, o capítulo cinco apresenta as conclusões obtidas com relação aos resultados obtidos e as recomendações feitas para trabalhos deste tipo e para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Definições Iniciais

Para a execução deste trabalho são necessários alguns conceitos e definições básicas para o correto entendimento dos temas apresentados. As definições iniciais aqui tratadas serão a de linguagem de programação, foco das atividades empreendidas, e a definição das duas linguagens de programação aqui comparadas, tratando de classificá-las e mostrar seus componentes básicos. A seguir, o conceito de otimização será abordado, além da classificação de seus métodos (exatos e heurísticos) e posterior apresentação do algoritmo genético, um dos métodos heurístico utilizados na área de otimização, mostrando sua definição e um exemplo do algoritmo mais básico.

2.1.1 Linguagem de Programação

As instruções feitas por programadores para que o computador execute alguma tarefa são realizadas utilizando-se uma linguagem de programação. Esta linguagem é responsável por traduzir uma determinada ação em códigos de máquina, algo compreensível pelo computador. Deitel (2006) mostra que algumas LPs são facilmente compreensíveis, enquanto outras requerem passos intermediários de tradução. Devido a isto, o autor divide as LPs em três tipos gerais:

1. Linguagem de máquina: É a linguagem "natural" de um computador e como tal é definida pelo seu design de hardware. Essas linguagens consistem geralmente em strings de números que instruem o computador a realizar suas operações mais elementares uma de cada vez. Por serem consideradas linguagens dependentes de máquina, uma linguagem de máquina particular pode ser utilizada em apenas um tipo de computador.
2. Linguagem assembly: Como as linguagens de máquina são muito lentas e propensas a erro para a maioria dos programadores, estes começaram a utilizar abreviações em inglês para representar as operações elementares. Foram estas abreviações que formaram a base das linguagens assembly. Para que os primeiros programas de linguagem assembly fossem convertidos em linguagem de máquina, são utilizados os programas tradutores chamados assemblers.

3. Linguagem de alto nível: Para que o processo de programação não exigisse muito tempo do programador, foram desenvolvidas as linguagens de alto nível, em que instruções únicas poderiam ser escritas para realizar tarefas substanciais. Seus programas tradutores, chamados de compiladores convertem os programas de linguagem de alto nível em linguagem de máquina. Como este processo de compilação pode consumir uma grande quantidade de tempo de processamento, programas interpretadores foram desenvolvidos para executar programas de linguagem de alto nível diretamente, embora muito mais lentamente. Os interpretadores são populares em ambientes de desenvolvimento de programa em que novos recursos vão sendo adicionados, e os erros, corrigidos. No final, quando o programa já está completamente desenvolvido, uma versão compilada pode ser produzida para executar mais eficientemente.

2.1.1.1 C++

Aqui será apresentado o C++ enquanto linguagem de programação. Inicialmente são dados seus componentes históricos e definições de suas características básicas. Por fim é dada a estrutura geral de uma função nesta linguagem.

O C++ é uma linguagem de programação de alto nível, derivada da linguagem C. Em seu histórico, Savitch (2004) afirma que ela foi desenvolvida por Bjarne Stroustrup, dos AT&T Bell Laboratories, na década de 80. A maior parte da linguagem C é um subconjunto da C++, e, assim, muitos programas em C são programas em C++. A principal diferença entre estas é que a C++ possui recursos para classes e, portanto, pode ser usada para a programação orientada a objetos.

A programação orientada a objetos (Object-oriented programming – OOP) é uma técnica de programação que, por trabalhar com classes, é ideal para trabalhar com dados e algoritmos. A OOP possui como características encapsulamento, herança e polimorfismo. O encapsulamento é uma forma de ocultação de informação, ou abstração. A herança tem a ver com a escrita de código reutilizável e, por fim, o polimorfismo se refere à forma pela qual um único nome pode ter múltiplos significados no contexto da herança. Além disso, suas variáveis, expressões e atribuições são similares às da maioria das outras linguagens de finalidade geral. (SAVITCH, 2004)

Mizrahi (2006) mostra que um programa em C++ consiste em uma ou várias funções, classes, e um conjunto de objetos que se comunica por meio de chamadas às funções membros. A forma geral de um função em C++ pode ser dada como ilustrado abaixo:

```
Tipo nomeFunc(declaração dos paramêtros)
{
        intrucao_1;
        instrução_2;
        ...
        instrução_n;
        return var_tipo;
}
```

Figura 1 - *Forma geral de uma função em C++. (MIZRAHI, 2006)*

2.1.1.2 Java

Aqui será apresentada a definição de Java enquanto linguagem de programação. Serão também serão abordados os seu componentes históricos e as partes que consistem os programas em Java.

Concebido pela Sun Microsystems e liberado para o público em 1995, o Java se baseou nas duas linguagens que eram, até então, as mais amplamente utilizadas no mundo, o C e o C++. Goslin (2000) apresenta a linguagem Java como sendo uma linguagem de alto nível de uso geral, baseada em classes e orientada a objetos. Deitel (2002) complementa a definição, afirmando que os programas em Java são constituídos basicamente destas classes.

Segundo Deitel (2011) um programa compilado em Java é executado por uma Máquina Virtual Java (Java Virtual Machine – JVM) – uma parte integrante do kit de desenvolvimento Java, e uma peça fundamental da plataforma. Uma máquina virtual, como a JVM, é uma aplicação que simula um computador, escondendo o sistema operacional e o hardware dos programas que interagem com ele. A vantagem é que se uma JVM é implementada em diferentes plataformas, os aplicativos que ela executa funcionará em todas elas. Sua principal desvantagem é que os programa interpretado pela JVM são mais lentos, quando comparados com programas desenvolvidos em ambientes puramente compilados. Isto é devido ao fato de que cada instrução do código intermediário deve ser lida, decodificada e traduzida para código de máquina, para que depois possa ser executada. Este processo ocasiona um atraso na execução do programa. (RANGEL, 2012)

Quanto à sua utilização, o Java é uma linguagem verdadeiramente portável, de modo que seja apropriado para a implementação de aplicativos baseados em diferentes ambientes e,

em especial, na Internet. Devido a isto, possui recursos como strings, imagens gráficas, componentes de interface gráfica com o usuário, tratamento de exceções, multithreading, multimídia, processamento de arquivos, processamento de banco de dados, redes clientes/servidor e estrutura de dados pré-empacotadas. (DEITEL, 2002)

2.1.2 Otimização

Este item irá apresentar a definição do termo otimização, como ela é feita, a modelagem de problemas deste tipo e, por fim, irá apresentar os dois métodos de resolução utilizáveis para a otimização.

A otimização pode ser definida como a ciência da determinação das melhores soluções de um dado problema matemático. Ela envolve o estudo de critérios, restrições, a determinação de um algoritmo como método de solução e o estudo da estrutura de tais métodos, além de sua experimentação em situações de teste e em problemas da vida real. (FLETCHER, 2000)

Para se otimizar um problema de situação real, Taha (2008) afirma que é necessário representá-lo utilizando-se de modelos hipotéticos. Um modelo hipotético é uma abstração do mundo real considerado. Ele deve apresentar as variáveis dominantes que controlam o comportamento do sistema real e, dessa forma, expressar de forma tratável as funções matemáticas que representam o comportamento do ambiente considerado. Winston (2004) completa esta afirmação dizendo que um modelo de otimização busca achar os valores das variáveis de decisão que otimizam (maximizam ou minimizam) a função objetivo, e que estejam dentro dos valores que satisfazem as restrições dadas.

Quanto aos seus métodos de resolução, a otimização pode ser realizada via métodos exatos ou métodos heurísticos.

2.1.2.1 Métodos Exatos

Uns dos métodos de resolução de problemas de otimização são os métodos exatos. Rothlauf (2011) trata os métodos exatos como métodos de otimização que garantem a obtenção da solução ótima de um determinado problema. Para que isto aconteça, Hooker (2015) afirma que este método geralmente se utiliza de ramificações e outras formas de busca exaustivas para se chegar à solução ótima.

Geralmente, os métodos exatos são os escolhidos para tratarem de casos em que as tentativas necessárias para se resolver o problema crescem de forma polinomial, de acordo com o tamanho deste problema. Casos contrários, em que a complexidade do problema cresce exponencialmente e estes se tornam intratáveis, não conseguem ser resolvidas via métodos exatos. Isto porque tanto o tempo de execução quanto o uso da memória se tornariam demasiadamente grandes. (ROTHLAUF, 2011)

A fim de exemplificar alguns dos principais métodos exatos de programação, a tabela 02 abaixo é dada:

Método Exato	Descrição
Simplex	O método simplex é um procedimento algébrico que visa a solução de problemas de programação linear. Devido a isto, ele visa buscar soluções em que vários aspectos (restrições) devem ser considerados. Sendo assim, dado um determinado problema, são estabelecidas inequações que traduzam as restrições apresentadas. A partir daí são atribuídos valores às variáveis que se deseja otimizar, até se obter o resultado de forma mais rápida possível.
Branch and Bound	O conceito básico por trás deste método é o de *dividir* e *conquistar*. Quando um problema dado e demasiadamente grande e muito difícil para ser resolvido diretamente, ele é dividido em subproblemas cada vez menores, até que esses subproblemas possam ser resolvidos. A divisão (branch) é feita dividindo-se todo o conjunto de possíveis soluções em menores subconjuntos. A conquista é feita limitando-se (bounding) o quão boa a solução dentro do subconjunto pode ser, e depois descartando aquelas que não possuem a solução ótima para o problema.
Programação Dinâmica	É uma técnica matemática utilizada para se criar uma sequência de decisões inter-relacionadas. Sendo assim, é um procedimento sistemático para se determinar a combinação ótima de decisões a serem tomadas.

Tabela 2 - *Alguns dos principais métodos exatos. Adaptado de Hillier (2009)*

2.1.3 Métodos Heurísticos

Outros métodos para se tratar de problemas de otimização são os métodos heurísticos. Hillier (2010) afirma que um método heurístico de otimização é um procedimento que provavelmente vai encontrar uma excelente solução viável, mas não necessariamente uma solução ótima para um dado problema específico. Devido a isto, não se pode dar nenhuma garantia sobre a qualidade da solução obtida, apesar de que, um método heurístico bem elaborado, é capaz de fornecer uma solução que se encontra pelo menos próxima da ótima. Em sua execução são feitas iterações que envolvem a busca por uma nova solução que, eventualmente, pode ou não ser melhor que a melhor solução encontrada na iteração

anterior. Quando o algoritmo termina após um tempo determinado, a solução por ele fornecida é a melhor que foi encontrada durante qualquer das iterações.

Além disso, Linden (2008) diz que quando se trata de problemas de grande porte, os tempos de execução são consideravelmente menores que os métodos exatos.

Como exemplo de heurísticas, Hillier (2009) apresenta cinco dos principais métodos, como brevemente descritos na Tabela 03 abaixo:

Método Heurístico	Descrição
Busca Tabu (Tabu Search)	Método de otimização que, a partir de uma solução inicial, busca avançar para a próxima solução (melhor que a anterior) na sua vizinhança, até que seja atingido um critério de parada definido. É um método de otimização global, por isso, utiliza-se de técnicas para escapar de ótimos locais.
Recozimento Simulado (Simmulated Annealing)	Utilizado como uma metáfora de um processo térmico, o recozimento simulado utiliza de iterações para sair de uma solução atual para uma próxima de acordo com sua função objetivo e uma variável definida T. Quanto maior o valor de T, maior a aleatoriedade da próxima solução escolhida. Conforme T é decrescido, o algoritmo começa a convergir para uma solução ótima.
Algoritmo Genético (Genetic Algorithm)	Análogo ao processo genético da seleção natural, o algoritmo genético busca através de uma população inicial de resultados, obter novas gerações com valores melhores. Isto é feito através de operadores que atuam realizando a seleção, recombinação e mutação das possíveis soluções encontradas. Assim, a cada iteração, uma população possivelmente melhor é esperada.
Colônia de Formigas (Ant Colony)	Inspirado na observação do movimento das formigas ao saírem de sua colônia para buscarem comida, que deixam feromônios para marcarem a direção da trilha, o método percorre possíveis caminhos que representam o problema a ser resolvido. Quanto menor é este caminho, mais importância ele recebe (feedback positivo), e após determinado número de iterações uma possível solução ótima é encontrada.
Busca Dispersa (Scatter Search)	É um método evolutivo de otimização, que busca através de uma população inicial de soluções buscar novas populações melhores. Diferentemente dos outros métodos evolutivos, os operadores deste métodos são limitados quanto à escolha aleatória das soluções, trabalhando somente em regiões pré-determinadas a cada iteração.

Tabela 3 - *Alguns dos principais métodos heurísticos. Adaptado de Hillier (2009)*

2.1.4 Algoritmo Genético

Os algoritmos genéticos (AG) são técnicas heurísticas de otimização global. Podem ser definidos como algoritmos de busca baseados nos mecanismos de seleção natural e genética. Neles, populações de indivíduos são criadas e submetidas aos operadores genéticos: Seleção, Recombinação (crossover) e Mutação. Após serem aplicados estes operadores, as populações são levadas à fase de avaliação, que submete estes indivíduos a

um processo de evolução natural. Assim, eles podem novamente, através dos operadores, gerar um indivíduo que caracterizará uma boa solução (que poderá ser a melhor possível) para o problema determinado. Assim, após determinado número de iterações, a sobrevivência dos melhores indivíduos através destes operadores, formam a estrutura de busca do AG. (LINDEN, 2008)

Tanomaru (1995) afirma que os AGs pertencem à classe dos métodos probabilísticos de busca e otimização, embora não trabalhem com buscas aleatórias. Na verdade, o AG tenta direcionar a sua busca para regiões do espaço de soluções onde é provável que os pontos ótimos estejam. Além disso, o fato de ser um método de otimização global favorece o uso dos AGs, pois são mais difíceis de ficarem presos em máximos locais e podem, de forma estruturada, chegar a um máximo global.

Linden (2008) resume o esquema de um AG em forma algorítmica de acordo com a Figura 02:

```
T:=0                                    //Inicializa-se o contador de tempo
Inicializa_População P(0)               //Inicializa-se a população de forma aleatória
Enquanto não terminar faça              //condição de término (tempo, avaliação, etc.)
        Avalia_população P(t)           //Avalia-se a população
        P':=Seleção_Pais P(t)           //Seleciona-se nova população que gerará uma nova
        P'=Crossover                    //Aplica-se o operador de crossover
        P'=Mutação                      //Aplica-se o operador de mutação
        Avalia_População P'             //Avalia-se a nova população
        P(t+1)=Seleção_sobreviventes P(t),P'    //Seleciona-se sobreviventes da nova geração
T:=T+1                                  //Incrementa-se o contador de tempo
Fim enquanto
```

Figura 2 - *Algoritmo de um AG básico. Adaptado de Linden (2008)*

2.2 Ferramentas de Programação

As ferramentas de programação são aplicativos simples e pequenos que realizam algum papel chave na programação de um programa (criação, depuração, manutenção, etc.). Elas geralmente são consolidadas em aplicativos mais robustos, que integram todas as funcionalidades em um único lugar. Estes são chamados de Ambientes de Desenvolvimento Integrado (Integrated Development Environment – IDE). Alguns exemplos de IDEs para C++ e Java são apresentados a seguir. Para uma lista mais completa de IDEs para estas linguagens é recomendada a leitura de Satav (2011).

2.2.1 Dev-C++

O Dev-C++ é um IDE para se programar em C e C++. É compatível somente com o Windows. De acordo com o seu site oficial (http://www.bloodshed.net/devcpp.html), é possível o download de pacotes e bibliotecas que aumentam o escopo e funcionalidade do IDE, sendo então possível a utilização de gráficos, compressão, animação, suporte a som, entre outros. Seu código foi desenvolvido em Delphi. (BLOODSHED, 2015)

A sua última versão é a 5 (em fase beta), lançada em 21 de Fevereiro de 2005.

2.2.2 Code::Blocks

O Code::Blocks, de acordo com seu site oficial, é um IDE livre capaz de trabalhar com as linguagens C, C++ e Fortran. Foi desenvolvido para ser capaz de se estender através de plugins e ser completamente configurável. É um IDE multiplataforma e, por ser completamente voltado para plug-ins, qualquer tipo de funcionalidade pode ser adicionada a ele, como os módulos de compilação e debugging que já são, via padrão, fornecidos dentro do IDE. Seu código foi desenvolvido em C++. (CODEBLOCKS, 2015)

A sua última versão é a 13.12, lançada em 27 de Dezembro de 2013.

2.2.3 Eclipse

Eclipse é uma comunidade para que indivíduos e organizações possam colaborar na criação de software livre. O foco de seus projetos é de construir uma plataforma de desenvolvimento aberta compreendida por inúmeras ferramentas para a construção, implantação e manutenção de softwares. (ECLIPSE, 2015)

De acordo com seu site oficial, o IDE do Eclipse é livre, multiplataforma e, assim como os outros, aceita facilmente a inserção de plug-ins em seu sistema. Ele também possui plataformas para outras aplicações, a saber: Plataforma Rich Client (RCP), Plataforma de Servidor, Plataforma de ferramentas Web (WTP) e Plataforma de Modelagem.

A sua última versão é a 4.4.2 (Luna SR2), lançada em 27 de fevereiro de 2015.

2.2.4 NetBeans

O IDE NetBeans fornece em seu pacote padrão a análise de código e editores para se programar com o Java 8 e outras linguagens e ferramentas, como o Maven, JavaScript,

HTML5, PHP e C/C++. Ele é um software livre e multiplataforma, sendo capaz de criar aplicações para desktop, mobile e web. (NETBEANS, 2015)
Sua última versão é a 8.0.2, lançada em 28 de Novembro de 2014.

2.3 Comparação entre Linguagens, Ferramentas e Algoritmos

Existem na literatura trabalhos que tratam de comparar diferentes linguagens de programação, diferentes ferramentas (como as IDEs) e diferentes algoritmos entre si.

A exemplo da comparação entre linguagens de programação, destaca-se Prechelt (2000) que compara, de forma empírica, sete linguagens de programação, com relação ao tempo de execução, alocação de memória, estrutura dos programas, entre outros; Tang (2013) que utiliza várias linguagens de programação para resolver uma equação em métodos iterativos lineares; e Bal (1992), que faz um estudo comparativo de cinco linguagens de programação paralelas. Com relação à comparação de ferramentas, Satav (2011) realiza um estudo comparativo e uma análise crítica a respeito de várias IDEs nas linguagens C/C++ e Java, para um desenvolvimento de softwares ideal; e Parreira (2001) realiza uma comparação de tempos de execução de algoritmos maxmin em diferentes compiladores. Por fim, na comparação de algoritmos, destacam-se Falcone (2004), que faz um estudo comparativo entre algoritmos genéticos e evolução diferencial para otimização de um modelo de cadeia de suprimento; Lim (2013), que compara a performance entre algoritmos genético, de evolução diferencial e de otimização por enxame de partículas em funções de teste; Ishibuchi (2006) que compara a performance entre algoritmos genéticos mono e multi objetivos; e Murata (1994) que mediu a performance de diferentes algoritmos genéticos em problemas de programação da produção em sistemas flow shop.

3. PLANEJAMENTO EXPERIMENTAL

Neste capítulo é descrito o procedimento metodológico empregado neste trabalho, enfatizando-se como foi realizado o planejamento pré-experimental e a descrição do experimento em si. Os dados coletados com o experimento serão depois utilizados na obtenção de resultados para a fundamentação do presente estudo.

A importância metodológica de um trabalho pode ser justificada pela necessidade de um fundamento científico adequado e pela busca da melhor abordagem para endereçar as questões da pesquisa (MIGUEL, 2012).

3.1 Planejamento Pré-Experimental

O planejamento pré-experimental visa caracterizar, definir um modelo e criar uma estratégia de ação a um experimento proposto. Montgomery (2001) mostra que quando este é bem planejado, pode melhorar o rendimento do processo experimental, além de reduzir a variabilidade dos dados e do tempo de desenvolvimento do experimento. Para isso, o autor sugere que um planejamento deste tipo deve detalhar os fatores e níveis que exercem influência na obtenção dos dados do experimento, as variáveis de resposta que o experimento irá resultar, os materiais e métodos a serem utilizados e por fim, o teste de hipóteses ao qual o experimento deverá passar para que este possa ser validado ou não.

3.1.1 Fatores, Níveis e Variáveis de resposta

Para a realização da análise de desempenho proposta foram definidos fatores e níveis que exercem influência no desenvolvimento e execução de um algoritmo genético. Neste experimento existirá apenas um fator: a combinação Linguagem de Programação + Ambiente de Programação. Quanto aos níveis, teremos:

- Nível 01: Linguagem C++, Ambiente Dev-C++;
- Nível 02: Linguagem C++, Ambiente Code::Blocks;
- Nível 03: Linguagem Java, Ambiente Eclipse;
- Nível 04: Linguagem Java, Ambiente NetBeans.

Tanto o fator, quanto os níveis foram considerados qualitativos, pois deseja-se apenas estabelecer a influência que estes exercem sobre o desempenho do AG.

Com relação às variáveis de resposta escolhidas para a análise de desempenho, serão considerados: tempo de execução do algoritmo, solução final encontrada e número de iterações necessárias para se atingir o valor mais próximo possível do ótimo conhecido dos problemas.

3.1.2 Materiais e Métodos

Para a análise de desempenho do AG serão empregados alguns materiais e métodos para que haja consistência científica nos dados obtidos. Como material, tem-se:

- Computador: O computador escolhido para que sejam executados os testes para a análise de desempenho trabalhará com uma máquina virtual executando a versão 2002 do Microsoft Windows XP (Service Pack 3). Sua configuração será um processador i7-3520M 2.9 GHz e 2 gb de memória RAM.

Na execução do algoritmo, os parâmetros a serem utilizados foram definidos de acordo com uma análise semelhante, realizada por Lim (2013). A tabela 04 apresenta estes parâmetros definidos. Eles se justificam por serem suficientes para que o algoritmo possa ter seu desempenho avaliado, já que permitem sua execução por completo, além de permitirem a obtenção do mínimo das funções de teste utilizadas, já que muitas exigem bastante de parâmetros como o tamanho da população e número de gerações. Além disso, eles devem ser pré-definidos para provarem a diferença de performance entre cada um dos níveis, já que há inúmeras diferenças decorrentes da interpretação do algoritmo entre eles.

Parâmetro	Valor
Número de iterações	2000
Tamanho da população	40
Número de dimensões	2
Probabilidade de crossover	0.07
Probabilidade de mutação	0.01
Número de execuções	10

Tabela 4 - *Parâmetros de execução do AG. Adaptado de Lim (2013).*

Com relação à escolha das funções de teste a serem utilizadas neste experimento, foram escolhidas cinco que pudessem representar características diversas das funções objetivos comumente usadas. Sendo assim, foram escolhidas três funções unimodais (De Jong,

Easom, Rosenbrock) e duas funções multimodais (Goldstein-Price, Ackley). Optou-se também por trabalhar com apenas duas variáveis (*x*,*y*) dentro das funções tendo em vista que a maioria delas, por natureza, são limitadas a estas duas variáveis (Easom, Goldstein-Price, Ackley). Devido a isto as outras funções N-dimensionais (De Jong, Roenbrock) foram ajustadas. A fim de caracterizar individualmente estas funções, elas são listadas a seguir:

3.1.2.1 Função De Jong:

Também conhecida como Função Esférica, é uma das mais simples funções utilizadas para testes de desempenho. É uma função contínua, convexa e unimodal. (MOLGA, 2005)

A figura 3 apresenta seu gráfico, bem com sua fórmula, os pontos de mínimo global (*f(x,y)=0*) e o domínio de busca comumente usado nos testes.

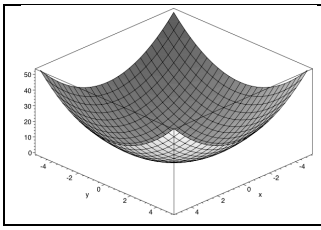
	Função De Jong
	$f(x,y) = x^2 + y^2$
	$f(x,y) = f(0,0) = 0$
	$-5.12 < x,y < 5.12$

Figura 3 - *Características da Função De Jong. Adaptado de Molga (2005).*

3.1.2.2 Função Easom:

É uma função de teste unimodal. Seu mínimo global possui uma área relativamente pequena quando comparada ao domínio de busca. (MOLGA, 2005)

A figura 4 apresenta seu gráfico, bem com sua fórmula, os pontos de mínimo global (*f(x,y)=-1*) e o domínio de busca comumente usado nos testes.

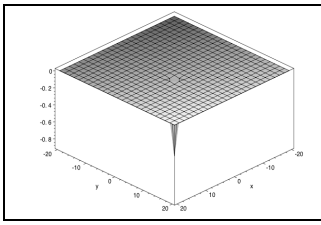
	Função Easom
	$f(x,y) = -\cos(x)\ \cos(y)\ \exp\left(-((x-\pi)^2 + (y-\pi)^2)\right)$
	$-100 \leq\ x,y\ \leq 100$
	$f(\pi,\pi) = -1$

Figura 4 - *Características da Função Easom. Adaptado de Molga (2005).*

3.1.2.3 Função Rosenbrock:

É um problema clássico de otimização. É também conhecida como Função banana ou Segunda Função de De Jong. Seu ótimo global se encontra dentro de um vale grande, estreito e parabólico.

Price (2005) define esta função como sendo unimodal e não-convexa. A figura 5 apresenta seu gráfico, bem com sua fórmula, os pontos de mínimo global (*f(x,y)=0*) e o domínio de busca comumente usado nos testes.

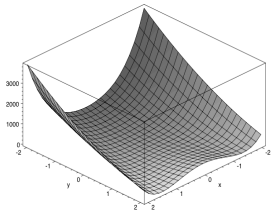
	Função Rosenbrock
	$f(x,y) = 100(y - x^2)^2 + (x-1)^2$
	$f(1,1) = 0$
	$-30 \leq x, y \leq 30$

Figura 5 - *Características da Função Rosenbrock. Adaptado de Molga (2005) e Price (2005).*

3.1.2.4 Função Goldstein-Price:

É uma função de teste para otimização global. Além disso, é uma função multimodal, com vários pontos de mínimo. A figura 6 apresenta suas principais características.

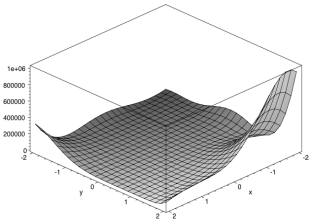
	Função Goldstein-Price
	$f(x,y) = \left(1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\right)$ $\left(30 + (2x-3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)\right)$
	$f(0,-1) = 3$
	$-2 \leq x, y \leq 2$

Figura 6 - *Características da Função Goldstein-Price. Adaptado de Molga (2005).*

3.1.2.5 Função Ackley:

Price (2005) afirma que a Função Ackley é uma das mais citadas funções de teste multimodais. Além disso, é uma função contínua. Foi obtida modulando uma função exponencial com um cosseno de amplitude moderada.

A figura 7 apresenta seu gráfico, bem com sua fórmula, os pontos de mínimo global (*f(x,y)=0*) e o domínio de busca comumente usado nos testes.

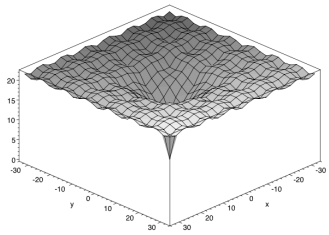
	Função Ackley
	$f(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right)$ $-\exp\left(0.5(\cos(2\pi x)+\cos(2\pi y))\right)+e+20$
	$f(0,0) = 0$
	$-32 \leq x, y \leq 32$

Figura 7 - *Características da Função Ackley. Adaptado de Molga (2005).*

3.1.3 Teste de Hipóteses

Como se deseja testar o desempenho dos diferentes níveis estabelecidos no planejamento pré-experimental, o teste de hipóteses visa testar as duas hipóteses possíveis para este problema: A hipótese nula (H_O) e a hipótese alternativa (H_1), a saber:

$$H_O: Não\ há\ diferença\ de\ desempenho\ entre\ os\ níveis$$
$$H_1: Há\ diferença\ de\ desempenho\ entre\ os\ níveis$$

4. RESULTADOS E ANÁLISES

Este capítulo apresenta os resultados obtidos pelo experimento e, em seguida, é feita uma análise dos dados para se responder à pergunta chave deste trabalho.

Os resultados obtidos pela execução do experimento são resumidos nos anexos I-V. Estes anexos apresentam, por função, cada uma das execuções feitas nos quatro diferentes IDEs, mostrando o menor resultado obtido, a geração em que este foi encontrado e os tempos acumulados e parciais das execuções.

Somente com estes resultados a seguinte análise prévia pode ser feita:

1. Com relação à função DeJong, todos os IDEs conseguiram encontrar o ponto de mínimo ótimo em suas execuções. Além disso, o tempo acumulado dos ambientes em C++ foi superior em relação aos ambientes em Java e, entretanto, os ótimos encontrados nos ambientes em Java ocorreram em gerações superiores, se comparados aos dos ambientes em C++;
2. Com relação à função Rosenbrock, nenhuma IDE atingiu o ótimo absoluto;
3. Com relação à função Goldstein-Price, somente uma execução no ambiente Dev-C++ obteve o ponto de mínimo ótimo.
4. Com relação à função Ackley, o ambiente Dev-C++ foi o que mais conseguiu chegar ao ponto de mínimo ótimo.
5. Com relação à função Easom, somente o ambiente Dev-C++ conseguiu obter o ponto de mínimo ótimo. Além disso, os ambientes em Java tiveram um tempo acumulado superior aos encontrados nos ambientes em C++.

Esta mesma análise pode ser verificada quando se observa o anexo VI, que demonstra os valores mínimos, médios e máximos das métricas utilizadas, para cada um dos ambientes estudados.

Para uma análise mais detalhada dos resultados, a fim de comprovar experimentalmente as hipóteses levantadas no teste de hipóteses, foram necessários testes mais completos. Para o caso das métricas tempo de execução e valor mínimo encontrado (aqui referenciadas como TEMPO e MÍNIMO) utilizou-se a técnica estatística ANOVA, que trata dados contínuos a fim de se verificar se há alguma diferença significativa entre eles. No caso da métrica número de iterações para se atingir o melhor resultado possível (referenciada como GERAÇÃO), foi necessária a utilização do Teste de Mood, que faz

uma análise não-paramétrica dos dados, neste caso discretos, utilizando-se as medianas ao invés das médias. Os testes foram realizados no software estatístico Minitab® 17.

A tabela 5 resume os p-values encontrados nos testes ANOVA e de Mood:

	DeJong	Rosenbrock	Goldstein-Price	Ackley	Easom
MÍNIMO	-	0.7918	0.1111	**<0.0001**	**0.0033**
GERAÇÃO	**<0.000**	0.776	0.133	**0.015**	0.261
TEMPO	0.4095	0.4072	**<0.0001**	**<0.0001**	0.187

Tabela 5 - *P-values das métricas em relação às funções. (Fonte: O Autor)*

De acordo com a Tabela 5, percebe-se então, em alguns casos, para o nível de significância utilizado ($\alpha = 0.5$), que há diferença significativa entre as médias/medianas das métricas utilizadas. É interessante então procurar onde estas diferenças estão ocorrendo. (OBS.: No caso da função DeJong, métrica MÍNIMO, não é possível realizar o teste ANOVA, tendo em vista que todos os IDEs obtiveram resultados iguais, indicando, portanto, que não há diferença significativa entre eles).

Nos testes feitos utilizando a ANOVA que obtuveram p-value < 0.05, o Teste de Fisher foi utilizado para se descobrir os casos em que estas diferenças significativas ocorrem. Seus resultados podem ser verificados nos anexos VII-X. A tabela 06 resume as descobertas obtidas com o teste.

Função e Métrica	Diferença observada
Goldstein - TEMPO	Há evidências estatísticas de que na métrica TEMPO o IDE Eclipse se difere dos demais, sendo ele o IDE de pior desempenho.
Ackley - MÍNIMO	Há evidências estatísticas de que na métrica MÍNIMO o IDE Code::Blocks possui o pior desempenho.
Ackley - TEMPO	Há evidências estatísticas de que na métrica TEMPO o IDE Eclipse apresentou os piores resultados com relação ao tempo.
Easom - MÍNIMO	Há evidências estatísticas de que há diferença na métrica MÍNIMO entre o IDE NetBeans apresenta uma diferença significativa dos demais.

Tabela 6 - *Diferenças observadas através do Teste de Fisher. (Fonte: O Autor)*

No caso da métrica GERAÇÃO, onde foram utilizados os Testes de Mood (vide anexos XI-XII), pode-se concluir para as funções DeJong e Ackley (que possuem p-value < 0.05), que há evidências estatísticas suficientes de que os IDEs tem um efeito nesta métrica.

Com esta análise realizada, podem ser feitas as conclusões finais deste trabalho, a fim de, finalmente, responder à pergunta chave deste trabalho.

5. CONCLUSÕES E RECOMENDAÇÕES

Neste capítulo, é apresentado um resumo das principais considerações advindas da realização deste trabalho, e em seguida, são apontadas as recomendações para a realização de trabalhos futuros.

5.1 Conclusões

Relembrando as hipóteses levantadas no planejamento experimental (H_O – Não há diferença de desempenho entre os níveis, e H_1 – Há diferença de desempenho entre os níveis), pode-se concluir que a hipótese nula H_O pode ser descartada, ou seja, os resultados mostraram que existem sim diferenças entre os IDEs e, consequentemente, entre as linguagens de programação aqui analisadas. Entretanto, como os resultados entre os diferentes problemas (funções) obtiveram resultados diferentes com relação ao testes estatísticos, não se pode afirmar com clareza qual linguagem (ou IDE) é melhor que a outra, pois percebe-se que diferentes tipos de problemas resultam em diferentes desempenhos dos níveis aqui estabelecidos. Um exemplo claro disto são as funções Rosenbrock e Ackley, que apresentaram comportamentos completamente distintos com relação às métricas observadas. Na primeira função, observou-se, com 95% de confiança, que não há diferença entre os IDEs analisados. Já na segunda, também com 95% de confiança, observou-se que em todas as métricas há diferenças significativas entre os IDEs. Neste caso, em algumas das métricas os IDEs C++ apresentam resultados melhores, am outras, os IDEs Java apresentam vantagem.

Deste modo, a fim de responder a questão problema deste trabalho:

"Há diferença de desempenho entre diferentes linguagens de programação em diferentes ambientes de programação para solucionar problemas iguais de otimização com a heurística algoritmo genético?"

Pode-se afirmar, com alto grau de precisão, que existe sim diferença de desempenho entre as linguagens de programação e também entre os diferentes ambientes de desenvolvimento.

5.2 Recomendações

Como recomendações para futuros trabalhos no tema aqui estudado, pode-se listar: Utilizar o experimento em outros problemas (novas funções teste ou problemas reais), analisar o comportamento do experimento quando se aumentam as dimensões das funções de teste, utilizar mais LPs, realizar a comparação utilizando-se um algoritmo diferente do AG e, finalmente, otimizar o algoritmo utilizando técnicas mais avançadas de programação para que novos testes possam ser feitos.

6. ANEXOS

DeJong

C++ - Dev-C++

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.00000	151	0.189	0.189
2	0.00000	89	0.372	0.183
3	0.00000	76	0.547	0.175
4	0.00000	19	0.738	0.191
5	0.00000	6	0.900	0.162
6	0.00000	24	1.079	0.179
7	0.00000	166	1.225	0.146
8	0.00000	159	4.148	2.923
9	0.00000	21	4.324	0.176
10	0.00000	11	4.495	0.171

C++ - Code::Blocks

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.00000	1740	0.09724	0.097
2	0.00000	1135	5.839859	5.743
3	0.00000	197	5.892083	0.052
4	0.00000	277	5.994843	0.103
5	0.00000	1059	6.048553	0.054
6	0.00000	373	6.145593	0.097
7	0.00000	1281	6.256793	0.111
8	0.00000	913	6.382413	0.126
9	0.00000	721	6.607913	0.226
10	0.00000	831	6.706863	0.099

Java - Eclipse

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.0000	1979	0.06913	0.069
2	0.0000	1999	0.10789	0.039
3	0.0000	1999	0.14515	0.037
4	0.0000	1997	0.18712	0.042
5	0.0000	1987	0.23379	0.047
6	0.0000	1968	0.28269	0.049
7	0.0000	1977	0.33088	0.048
8	0.0000	1989	0.36231	0.031
9	0.0000	1999	0.39968	0.037
10	0.0000	1989	0.44252	0.043

Java - NetBeans

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.0000	1987	0.06455	0.065
2	0.0000	1998	0.10447	0.040
3	0.0000	1998	0.14904	0.045
4	0.0000	1990	0.196587	0.048
5	0.0000	2000	0.231357	0.035
6	0.0000	1963	0.271157	0.040
7	0.0000	1999	0.315797	0.045
8	0.0000	2000	0.362617	0.047
9	0.0000	1997	0.394607	0.032
10	0.0000	1992	0.425957	0.031

Anexo I - *Resultados: Função DeJong*

Rosenbrock

C++ - Dev-C++

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.95516	9	0.088	0.088
2	1.00000	2	0.266	0.178
3	0.35116	42	0.306	0.040
4	0.76773	1	0.365	0.058
5	0.00797	297	0.407	0.042
6	0.77545	38	0.474	0.067
7	0.03408	1804	0.511	0.037
8	0.77691	4	0.572	0.061
9	0.43258	9	0.601	0.029
10	0.05797	5	0.671	0.070

C++ - Code::Blocks

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.39989	6	0.023	0.023
2	0.42332	1	0.037	0.014
3	0.47158	19	0.054	0.016
4	0.01498	3	0.090	0.036
5	0.18981	1509	0.109	0.019
6	0.59535	1	0.132	0.024
7	0.17017	11	0.152	0.020
8	0.55797	40	0.183	0.031
9	0.04513	13	0.202	0.018
10	0.62964	34	0.224	0.023

Java - Eclipse

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.0951	1	0.17808	0.178
2	0.0009	312	0.29425	0.116
3	0.1685	95	0.42887	0.135
4	0.6782	1	0.57578	0.147
5	0.7408	82	0.73729	0.162
6	0.1599	295	0.87872	0.141
7	3.9907	1	1.00379	0.125
8	0.0098	1	1.16478	0.161
9	0.7539	152	1.32812	0.163
10	0.4241	192	1.47049	0.142

Java - NetBeans

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.0070	69	0.084606	0.085
2	0.1128	126	0.130353	0.046
3	0.0621	93	0.177217	0.047
4	0.1438	219	0.225087	0.048
5	0.8659	13	0.277873	0.053
6	0.0012	1	2.554397	2.277
7	0.0946	40	2.608826	0.054
8	2.8366	1	2.653887	0.045
9	0.3067	14	2.696771	0.043
10	0.9930	14	2.755537	0.059

Anexo II - *Resultados: Função Rosenbrock*

Goldstein-Price

C++ - Dev-C++

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	3.00000	493	0.040	0.040
2	3.25095	234	0.081	0.041
3	97.49296	560	0.129	0.049
4	34.02117	63	0.168	0.039
5	9.37328	49	0.209	0.041
6	3.11548	31	0.260	0.051
7	29.82611	630	0.296	0.035
8	3.00091	1184	0.337	0.042
9	3.00315	196	0.382	0.045
10	13.27426	1	0.431	0.049

C++ - Code::Blocks

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	3.46740	90	0.015	0.015
2	4.79226	6	0.035	0.020
3	30.70451	276	0.060	0.025
4	85.54067	83	0.091	0.031
5	3.45062	27	0.117	0.026
6	3.51144	65	0.146	0.029
7	30.30969	704	0.169	0.023
8	3.98409	19	0.198	0.028
9	30.05215	853	0.216	0.018
10	3.64225	56	0.254	0.038

Java - Eclipse

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	86.9651	1	0.16275	0.163
2	35.5490	1	0.23200	0.069
3	3.0854	20	0.31680	0.085
4	32.8238	17	0.45432	0.138
5	53.2824	3	0.57101	0.117
6	3.0012	985	0.67575	0.105
7	23.0610	5	0.75516	0.079
8	57.0338	1	0.81669	0.062
9	98.9033	161	0.87081	0.054
10	15.2907	13	0.96589	0.095

Java - NetBeans

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	3.4628	37	0.059992	0.060
2	3.0003	295	0.086402	0.026
3	34.3512	307	0.108162	0.022
4	8.5942	260	0.125642	0.017
5	8.7737	1	0.167642	0.042
6	5.9913	155	0.191672	0.024
7	3.7033	32	0.215712	0.024
8	15.8305	1	0.242712	0.027
9	35.8913	7	0.268862	0.026
10	3.0044	1686	0.308612	0.040

Anexo III - *Resultados: Função Goldstein-Price*

Ackley

C++ - Dev-C++

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.00000	397	0.100	0.100
2	0.00000	8	0.233	0.133
3	0.00009	435	0.325	0.092
4	0.00000	1902	0.386	0.062
5	0.00000	1010	0.453	0.067
6	0.00000	499	0.576	0.124
7	11.33014	1	0.617	0.041
8	12.28562	1	0.673	0.056
9	9.20454	1	0.723	0.050
10	14.21684	1	0.768	0.044

C++ - Code::Blocks

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	8.1288	1	0.052	0.052
2	15.92732	1	0.087	0.035
3	11.8666	1	0.131	0.044
4	17.2328	1	0.188	0.057
5	7.23739	1	0.246	0.058
6	13.83127	1	0.289	0.043
7	10.93114	1	0.348	0.059
8	0	1998	0.397	0.049
9	13.9625	1	0.441	0.044
10	18.86193	1	0.490	0.048

Java - Eclipse

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	2.5845	113	0.20034	0.200
2	2.6827	153	0.38693	0.187
3	0.0000	1234	0.52418	0.137
4	0.0000	1526	0.68982	0.166
5	2.5839	93	0.86352	0.174
6	2.5837	105	1.06928	0.206
7	2.5971	128	1.26177	0.192
8	2.6083	169	1.45682	0.195
9	0.0000	1684	1.60604	0.149
10	0.0000	1302	1.75055	0.145

Java - NetBeans

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	0.0000	1284	0.09082	0.091
2	0.0000	1795	0.14459	0.054
3	2.7169	213	0.21976	0.075
4	2.6698	159	0.282669	0.063
5	2.7273	167	0.333099	0.050
6	2.7666	184	0.407549	0.074
7	0.0000	1197	0.463989	0.056
8	2.6365	130	0.520379	0.056
9	2.7418	144	0.592079	0.072
10	2.5827	126	0.655529	0.063

Anexo IV - *Resultados: Função Ackley*

Easom

C++ - Dev-C++

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	-0.99999	544	0.397	0.397
2	-0.99974	1	0.737	0.340
3	-0.91047	640	0.845	0.108
4	-0.98472	94	1.392	0.547
5	-0.92637	1588	2.819	1.427
6	-1.00000	386	3.098	0.279
7	-0.48890	925	13.410	10.312
8	-0.17849	227	48.323	34.913
9	-0.30821	1	48.498	0.175
10	-0.90801	1	48.773	0.276

C++ - Code::Blocks

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	-0.80632	60	0.303	0.303
2	-0.99926	959	0.589	0.286
3	-0.67088	21	1.103	0.514
4	-0.9758	46	1.424	0.321
5	-0.8566	265	35.786	34.362
6	-0.88593	56	36.266	0.480
7	-0.87989	21	36.865	0.599
8	-0.81287	129	37.155	0.290
9	-0.92527	1	37.470	0.315
10	-0.97627	1	37.663	0.193

Java - Eclipse

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	-0.99851	1220	0.08280	0.083
2	-0.83919	1	0.31699	0.234
3	-0.96616	114	0.75685	0.440
4	-0.97019	16	53.35670	52.600
5	-0.0577	1	53.70380	0.347
6	-0.98049	2	54.08037	0.377
7	-0.34088	1	54.70637	0.626
8	-0.99467	1135	55.40605	0.700
9	-0.96139	1140	267.64851	212.242
10	-0.67942	21	268.07553	0.427

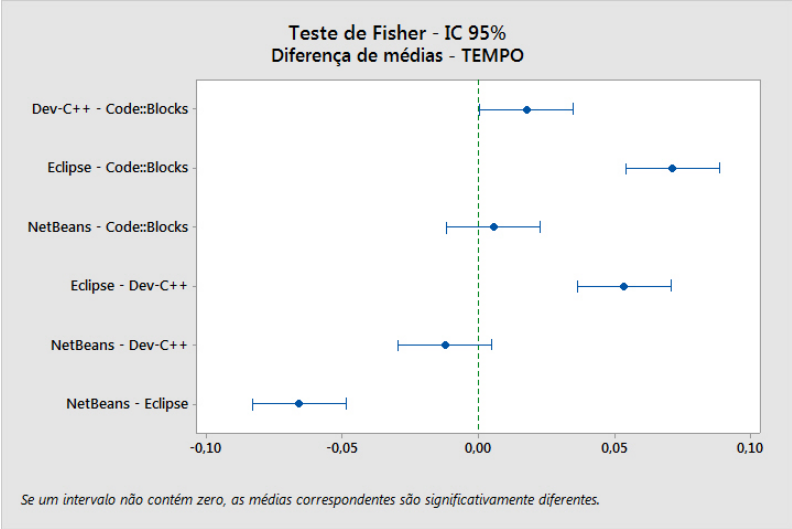
Java - NetBeans

Exec.	Menor Resultado	Geração	Tempo acumulado (s)	Tempo parcial (s)
1	-0.9976	98	1.5964	1.596
2	-0.3946	217	28.37173	26.775
3	-0.7437	945	28.87866	0.507
4	-0.0775	1	29.39411	0.515
5	-0.5213	1	29.836704	0.443
6	-0.0187	1	34.676205	4.840
7	-0.3041	16	35.335515	0.659
8	0.0000	1	158.986774	123.651
9	-0.6960	1	282.840882	123.854
10	-0.1677	1	757.277386	474.437

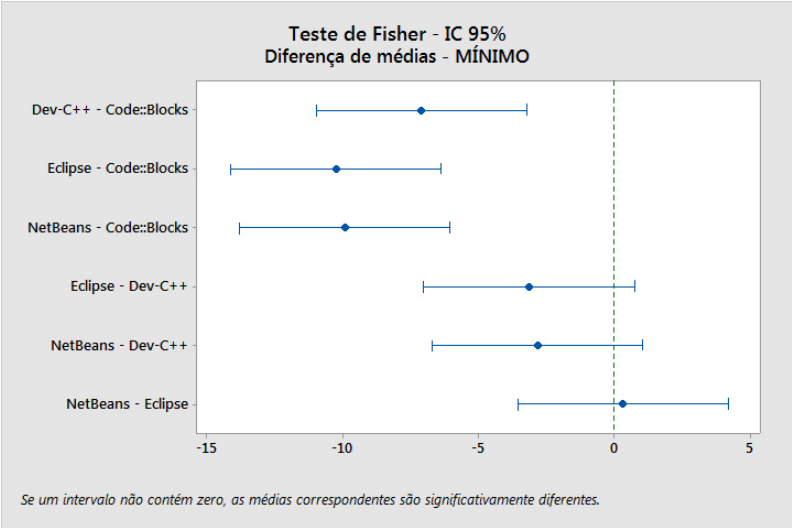
Anexo V - *Resultados: Função Easom*

			f(x)			Geração do ótimo encontrado			Tempo		
Função	**Ótimo**	**Método**	**Mínimo**	**Máximo**	**Médio**	**Mínimo**	**Máximo**	**Médio**	**Mínimo**	**Máximo**	**Médio**
De Jong	**0**	C++ - Dev-C++	0.00000	0.00000	0.00000	6	166	72	0.146	2.923	0.449
		C++ - Code::Blocks	0.00000	0.00000	0.00000	197	1740	853	0.052	5.743	0.671
		Java - Eclipse	0.00000	0.00000	0.00000	1968	1999	1988	0.031	0.069	0.044
		Java - NetBeans	0.00000	0.00000	0.00000	1963	2000	1992	0.031	0.065	0.043
Rosenbrock	**0**	C++ - Dev-C++	0.00797	1.00000	0.51590	1	1804	221	0.029	0.178	0.067
		C++ - Code::Blocks	0.01498	0.62964	0.34978	1	1509	164	0.014	0.036	0.022
		Java - Eclipse	0.00086	3.99068	0.70220	1	312	113	0.116	0.178	0.147
		Java - NetBeans	0.00122	2.83661	0.54236	1	219	59	0.043	2.277	0.276
Ackley	**0**	C++ - Dev-C++	0.00000	14.21684	4.70372	1	1902	426	0.041	0.133	0.077
		C++ - Code::Blocks	0.00000	18.86193	11.79798	1	1998	201	0.035	0.059	0.049
		Java - Eclipse	0.00000	2.68266	1.56401	93	1684	651	0.137	0.206	0.175
		Java - NetBeans	0.00000	2.76655	1.88415	126	1795	540	0.050	0.091	0.066
Easom	**-1**	C++ - Dev-C++	-1.00000	-0.17849	-0.77049	1	1588	441	0.108	34.913	4.877
		C++ - Code::Blocks	-0.99926	-0.67088	-0.87891	1	959	156	0.193	34.362	3.766
		Java - Eclipse	-0.99851	-0.05770	-0.77886	1	1220	365	0.083	212.242	26.808
		Java - NetBeans	-0.99757	-0.00003	-0.39212	1	945	128	0.443	474.437	75.728
Goldstein	**3**	C++ - Dev-C++	3.00000	97.49296	19.93583	1	1184	344	0.035	0.051	0.043
		C++ - Code::Blocks	3.45062	85.54067	19.94551	6	853	218	0.015	0.038	0.025
		Java - Eclipse	3.00122	98.90326	40.89958	1	985	121	0.054	0.163	0.097
		Java - NetBeans	3.00026	35.89129	12.26030	1	1686	278	0.017	0.060	0.031

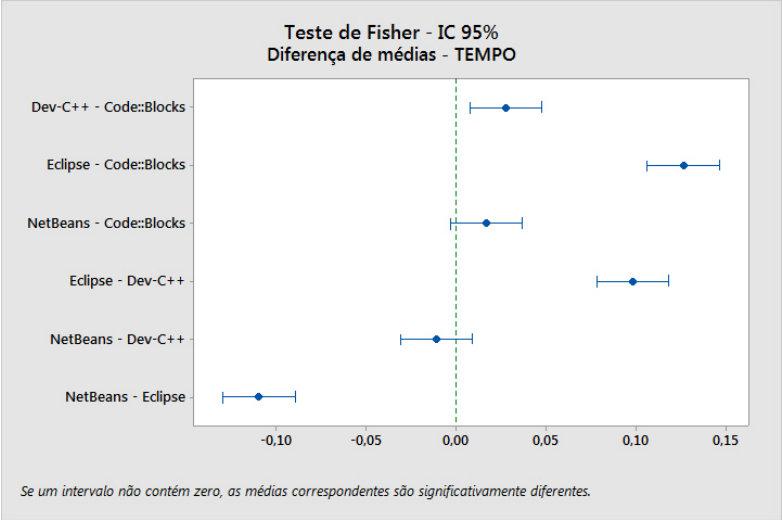
Anexo VI - *Valores mínimos, médios e máximos das métricas utilizadas, para cada um dos ambientes estudados.*



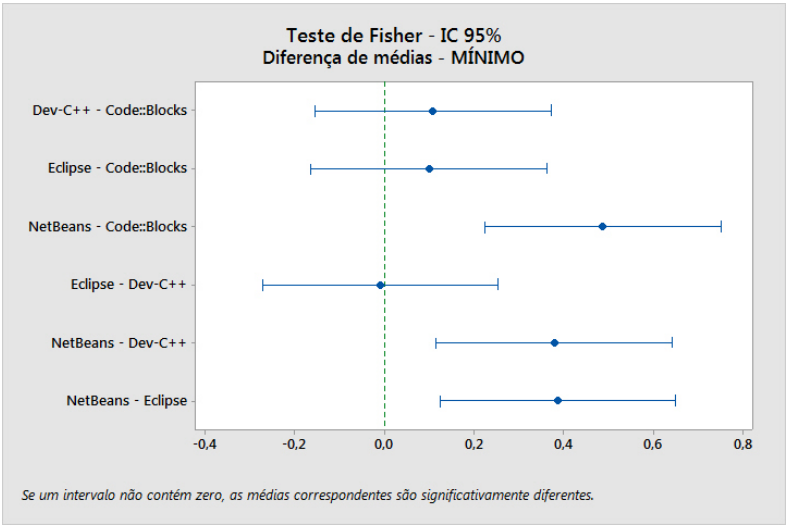
Anexo VII - *Teste de Fisher: Função Goldstein-Price vs. Tempo de execução*



Anexo VIII - *Teste de Fisher: Função Ackley vs. Valor mínimo encontrado*



Anexo IX - *Teste de Fisher: Função Ackley vs. Tempo de execução*



Anexo X - *Teste de Fisher: Função Easom vs. Valor mínimo encontrado*

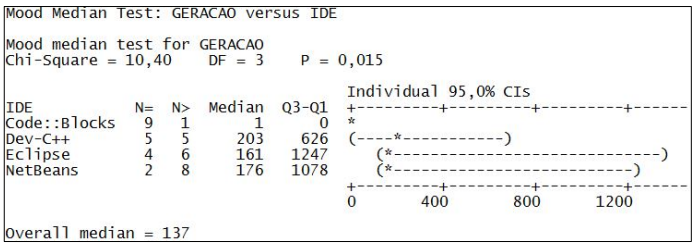
```
Mood Median Test GERACAO versus IDE

Mood median test for GERACAO
Chi-Square = 40,00    DF = 3    P = 0,000

                                    Individual 95,0% CIs
IDE           N=  N  Median  Q3-Q1  +---------+---------+---------+------
CodeBlocks  10   0     872    823        (------------)
Dev-C++       10   0      50    136  (-)
Eclipse        0  10    1989     21
NetBeans       0  10    1998     10
                                    +---------+---------+---------+------
                                    0        600      1200      1800

Overall median = 1852
```

Anexo XI - *Teste de Mood – Métrica Geração: Função DeJong*



```
Mood Median Test: GERACAO versus IDE

Mood median test for GERACAO
Chi-Square = 10,40    DF = 3    P = 0,015

                                          Individual 95,0% CIs
IDE           N=  N>  Median  Q3-Q1  +---------+---------+---------+------
Code::Blocks   9   1       1     0  *
Dev-C++        5   5     203   626  (----*-----------)
Eclipse        4   6     161  1247     (*-----------------------------)
NetBeans       2   8     176  1078     (*--------------------------)
                                      +---------+---------+---------+------
                                      0        400       800       1200

Overall median = 137
```

Anexo XII - *Teste de Mood – Métrica Geração: Função Ackley*

7. REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, Rogério Henrique C.; SILVA, Wilton Lacerda. Detecção de Falhas em Sistemas Automatizados com o OpenGL. ***Ciência & Desenvolvimento – Revista Eletrônica da FAINOR.*** vol. 1, no. 1, p.p. 10-11, 2008.

ARORA, P. K. Et al. Design of a Production System Using Genetic Algorithm, ***Procedia Technology.*** vol 14, p.p. 390-396, 2014.

BAL, Henri E. A comparative study of five parallel programming languages. ***Future Generation Computer Systems.*** vol. 8, no. 1, p.p. 121-135, 1992.

BELFIORE, Patrícia; FÁVERO, Luiz P. ***Pesquisa Operacional para Cursos de Engenharia.*** Vol. 1. Rio de Janeiro: Elsevier, 2013

BLOODSHED. **Dev-C++.** Disponível em: <http://www.bloodshed.net/devcpp.html>. Acesso em: 12 mar. 2015.

BOCK, Heiko. ***The Definitive Guide to NetBeans Platform.*** Apress, 2009.

BOUDREAU, Tim et al. ***NetBeans:*** The Definitive Guide. O'Reilly Media Inc., 2002.

BROOKSHEAR, J. Glenn. ***Ciência da Computação:*** Uma Visão. 7ª ed. São Paulo: Bookman, 2005.

BUDINSKY, Frank. ***Eclipse Modeling Framework:*** A Developer's Guide. Addison-Wesley Professional, 2004.

CHEN, Jian; TAN, Jun-shan. Extended Method of Generic Container Based on STL. ***Information Technology.*** vol. 6, p.p. 017, 2006.

CHEN, Zhixiong; MARX, Delia. Experiences with Eclipse IDE in programming courses. ***Journal of Computing Sciences in Colleges***. vol. 21, no. 2, p.p. 104-112, 2005.

CODEBLOCKS. **The open source, cross platform, free C, C++ and Fortran IDE.** Disponível em: <http://www.codeblocks.org>. Acesso em: 11 mar. 2015.

CONSTANTINO, Ademir Aparecido et al. ***Aplicação de Algoritmos Genéticos ao Problema de Cobertura de Conjunto.*** XXXV Simpósio Brasileiro de Pesquisa Operacional – SBPO Natal, 2003.

DEITEL, H. M.; DEITEL, P. J.. **Java: Como Programar.** 4ª ed. Porto Alegre: Bookman, 2002.

DEITEL, Harvey; DEITEL, Paul. ***C++:*** Como Programar. 5ª Edição. São Paulo: Pearson Prentice Hall, 2006

DEITEL, P.; DEITEL, H. ***Java:*** How to Program. 9ª ed. Prentice Hall, 2011.

DELMAN, Amy et al. ***Development of a System for Teaching C/C++ Using Robots and Open Source Software in a CS1 Course.*** In: Frontiers in Education: Computer Science & Computer Engineering – FECS Las Vegas, 2009.

ECLIPSE. **About the Eclipse Foundation.** Disponível em: <https://eclipse.org/org/>. Acesso em: 12 mar. 2015.

ENGELBRECHT, Andries P. ***Computational Intelligence:*** An Introduction. 2ª ed. John Wiley & Sons, 2007.

FAIRBARN, Andrew. ***O Manual Dev C++ E Wx Dev C++.*** Clube de Autores, 2008.

FALCONE, Marco Aurélio Guia. ***Estudo Comparativo Entre Algoritmos Genéticos e Evoluçao Diferencial para Otimizaçao de um Modelo de Cadeia de Suprimento Simplificada.*** Dissertação em Engenharia de Produção e Sistemas – PUC, Paraná. 2004.

FERREIRA, Déborah Mendes; ROSA, Lucas Pessoa. ***Utilização De Algoritmos Genéticos Para Sequenciamento De Partidas Em Aeroportos.*** Monografia de Bacharelado em Ciência da Computação – UNB, Brasília – DF, 2013.

FIORENTINO, Helenice O. et al. Multiobjective Genetic Algorithm Applied To Dengue Control. ***Mathematical Biosciences.*** vol 258, p.p. 77-84, 2014.

FLETCHER, R. ***Practical Methods of Optimization.*** 2ª ed., West Sussex: Wiley, 2000.

GALVÃO, Marcelo de Lima; LAMAR, Marcus Vinicius; TACO, Pastor Willy Gonzales. Desenho Automático De Mapas Octalineares De Rede De Transporte Público Utilizando Algoritmo Genético. ***Transportes***. vol. 22, no. 1, p.p. 21–30, 2014.

GEER, David. Eclipse becomes the dominant Java IDE. ***Computer***. vol. 38, no. 7, p.p. 16-18, 2005.

GOLFETO, Rodrigo Rabello; MORETTI, Antônio Carlos; SALLES NETO, L. L. ***Algoritmo Genético Simbiótico Aplicado Ao Problema De Corte Unidimensional.*** In: XXXIX Simpósio Brasileiro de Pesquisa Operacional – SBPO Fortaleza, 2007.

GOSLING, James et al. **The Java Language Specification**. 2ª ed. Addison-Wesley Professional, 2000.

HILLIER, Frederick S.; LIEBERMAN, Gerald J., ***Introdução à Pesquisa Operacional.*** 8ª ed., Porto Alegre: AMGH, 2010.

HILLIER, Frederick S.; LIEBERMAN, Gerald J.. **Introduction to Operations Research**. 9ª ed. McGraw-Hill, 2009.

HOOKER, J. N.. Toward unification of exact and heuristic optimization methods. **International Transactions in Operational Research**. vol. 22, no. 1, p.p. 19-48, 2015.

ISHIBUCHI, H.; NOJIMA, Yusuke; DOI, T., **Comparison between Single-Objective and Multi-Objective Genetic Algorithms: Performance Comparison and Performance Measures**. In: IEEE Congress on Evolutionary Computation – CEC Vancouver, vol., no., pp.1143,1150, 2006.

JUNIOR, Nelson Florêncio; GUIMARÃES, Frederico Gadelha. ***Problema 8-Puzzle:*** Análise da solução usando Backtracking e Algoritmos Genéticos. 2012.

LEE, Wonjae; KIM, Hak-Young. ***Genetic Algorithm Implementation In Python***. In: Fourth Annual ACIS International Conference on Computer and Information Science, 2005.

LIM, Seng Poh; HARON, H. ***Performance Comparison of Genetic Algorithm, Differential Evolution and Particle Swarm Optimization Toward Benchmark Functions.*** In: 2013 IEEE Conference on Open Systems – ICOS Kuching, 2013.

LINDEN, Ricardo. ***Algoritmos Genéticos:*** Uma Importante Ferramenta da Inteligência Computacional. 2ª ed. Rio de Janeiro: Brasport, 2008.

MIGUEL, Paulo Augusto Cauchick et al. ***Metodologia de Pesquisa em Engenharia de Produção e Gestão de Operações.*** 2ª ed. Rio de Janeiro: Elsevier, 2012.

MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C++**. 2ª ed. São Paulo: Pearson Prentice Hall, 2006.

MOLGA, Marcin; SMUTNICKI, Czesław. ***Test functions for optimization needs***. Disponível em <http://www.robertmarks.org/Classes/ENGR5358/Papers/functions.pdf> Acesso em: 11 dez. 2014.

MONTGOMERY, Douglas C. ***Design and Analysis of Experiments.*** 5ª ed. John Wiley and Sons, 2001.

MURATA, T.; ISHIBUCHI, H., ***Performance evaluation of genetic algorithms for flowshop scheduling problems.*** In: IEEE World Congress on Computational Intelligence, vol. 2, no., pp. 812,817, 1994.

NETBEANS. **NetBeans IDE Features.** Disponível em: <https://netbeans.org/features>. Acesso em: 12 mar. 2015.

OLIVEIRA, Douglas Coelho Braga; DA SILVA, Rodrigo Luis de Souza. Desenvolvimento de uma Biblioteca de Realidade Aumentada Orientada a Objetos baseada no ARToolkit. ***Relatórios Técnicos do DCC/UFJF***, 2013.

OLIVEIRA, Rômulo A.; JÚNIOR, Manoel F. M., MENEZES, Roberto Felipe A. Application Of Genetic Algorithm For Optimization On Projects Of Public Illumination. ***Electric Power Systems Research.*** vol 117, p.p. 84-93, 2014.

PARREIRA, Walteno Martins et al. Comparação de Tempo de Execução de Algoritmos MaxMin em Compiladores Diferentes, ***Intercurso Revista Científica.*** Vol. 09, no. 01, p.p 95-101, 2001

PIZZUTI, Clara. A Multiobjective Genetic Algorithm To Find Communities In Complex Networks. ***IEEE Transactions on Evolutionary Computation***, vol. 16, no. 3, p.p. 418-430, 2012.

PONNAMBALAM, S. G.; ARAVINDAN, P.; NAIDU, G. M. A Multi-Objective Genetic Algorithm For Solving Assembly Line Balancing Problem. ***The International Journal of Advanced Manufacturing Technology***, vol. 16, no. 5, p.p. 341-352, 2000.

PRECHELT, Lutz. An Empirical Comparison Of Seven Programming Languages. **Computer**. vol. 33, no. 10, p.p. 23-29, 2000.

PRESSMAN, Roger S. ***Software Engineering:*** A Practitioner's Approach. 6ª ed. Palgrave Macmillan, 2005.

PRICE, Kenneth; STORN, Rainer M.; LAMPINEN, Jouni A. ***Differential Evolution:*** A Practical Approach To Global Optimization. Springer Science & Business Media, 2006.

RANGEL, Francis; SILVA, Anderson Faustino. Máquina Virtual Java e a Otimização Inline: Um Estudo de Caso. ***Revista Tecnológica***. Maringá, vol. 21, p.p. 103-118, 2012.

ROTHLAUF, Franz. ***Design of modern heuristics:*** Principles and Application. Springer Science & Business Media, 2011.

SATAV, Sampada K.; SATPATHY, S. K.; SATAO, K. J. A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development, ***Universal Journal of Applied Computer Science and Technology***. Gwalior, vol. 01, no. 01, p.p. 09-15, 2011

SAVITCH, W. J. ***C++ Absoluto***. São Paulo: Addison Wesley, 2004.

SOUZA, Celso Correia et al. Utilização Dos Algoritmos Genéticos Como Ferramenta De Otimização Em Problemas De Roteirização. ***FACEF Pesquisa: Desenvolvimento e Gestão***. vol.15, no. 3 - p.p. 285-297, 2012.

TAHA, Hamdy A. ***Pesquisa Operacional.*** 8ª ed. São Paulo: Pearson Prentice Hall, 2008.

TANG, Tingting; LIU, Weiyun; MCDONOUGH J.M., Parallelization of Linear Iterative Methods for Solving the 3-D Pressure Poisson Equation Using Various Programming Languages. ***Procedia Engineering.*** vol. 61, no. 01, p.p. 136-143, 2013.

TANOMARU, Julio. Motivação, fundamentos e aplicações de algoritmos genéticos. In: **II Congresso Brasileiro de Redes Neurais**, 1995.

TIOBE. ***TIOBE Index for October 2014.*** Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acesso em: 5 nov. 2014.

VAN-ROY, Peter; HARIDI, Seif. ***Concepts, Techniques, And Models Of Computer Programming.*** MIT Press, 2004.

VERMA, Abhijit; AGGARWAL, Sachin; SRIVASTAVA, Siddhartha. Strain Measurement Using Image Processing. In: ***International Journal of Engineering Research and Technology.*** ESRSA Publications, 2013.

WINSTON, W. L.. ***Operations Research:*** Applications and Algorithms. 4ª ed. Belmont: Duxbury, 2004.

ZHANG, Liqiang et al. Multi-Objective Optimization Of Lithium-Ion Battery Model Using Genetic Algorithm Approach. ***Journal of Power Sources.*** vol. 270, p.p. 367-378, 2014