



**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
E INFORMÁTICA INDUSTRIAL**

MARLON HENRIQUE SCALABRIN

**MEGA BUSCA HARMÔNICA: ALGORITMO DE BUSCA
HARMÔNICA BASEADO EM POPULAÇÃO E IMPLEMENTADO EM
UNIDADES DE PROCESSAMENTO GRÁFICO**

DISSERTAÇÃO

CURITIBA

2012

MARLON HENRIQUE SCALABRIN

**MEGA BUSCA HARMÔNICA: ALGORITMO DE BUSCA
HARMÔNICA BASEADO EM POPULAÇÃO E IMPLEMENTADO EM
UNIDADES DE PROCESSAMENTO GRÁFICO**

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciências, do Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná – UTFPR. Área de Concentração: Engenharia de Computação.

Orientador: Prof. Dr. Heitor Silvério Lopes

CURITIBA

2012

Dados Internacionais de Catalogação na Publicação

- S281 Scalabrin, Marlon Henrique
Mega busca harmônica: algoritmo de busca harmônica baseado em população e implementado em unidades de processamento gráfico / Marlon Henrique Scalabrin. – 2012.
132 f. : il. ; 30 cm
- Orientador: Heitor Silvério Lopes.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2012.
Bibliografia: f. 83-89.
1. Análise harmônica. 2. Heurística. 3. Algoritmos. 4. Otimização combinatória. 5. Computação gráfica. 6. Arquitetura de computador. 7. Programação paralela (Computação). 8. Engenharia elétrica – Dissertações. I. Lopes, Heitor Silvério, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº: 590

“Mega Harmony Search: Algoritmo de Busca Harmônica Baseado em População e Implementado em Unidades de Processamento Gráfico”

por

Marlon Henrique Scalabrin

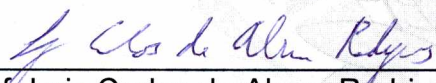
Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Engenharia de Computação, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Curitiba, às 09h do dia 31 de março de 2012. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:



Prof. Heitor Silvério Lopes, Dr.
(Presidente)



Prof. Wagner Rodrigo Weinert, Dr.
(IFPR – Câmpus. Paranaguá)



Prof. Luiz Carlos de Abreu Rodrigues, Dr.
(UTFPR)

Visto da coordenação:



Prof. Fábio Kurt Schneider, Dr.
(Coordenador do CPGEI)

Aos meus pais, Adivoncir e Mairi e aos meus irmãos por toda força, incentivo e compreensão.

Ao professor Heitor Silvério Lopes pela atenção e por acreditar em minha capacidade, me ensinando muito além dos limites deste trabalho.

Em especial, à minha noiva Ediane, pelo auxílio no desenvolvimento do trabalho e por sua atenção e amor, que me motivaram a prosseguir e concluir este trabalho.

AGRADECIMENTOS

A Deus, por me pegar no colo quando minhas pernas não possuíam mais forças para continuar.

Aos meus pais, pelo amor ilimitado em um mundo em que tudo é quantificável e limitado.

Aos meus irmãos, que me entendem, me consolam, me dão a direção, são minha base e minha melhor cumplicidade.

Aos meus familiares pelo apoio e palavras de incentivo.

À CAPES pela concessão de bolsa de mestrado, indispensável para a realização desta dissertação.

Ao professor Heitor pela atenção e dedicação prestadas no decorrer do trabalho e pela amizade conquistada.

Aos professores do Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial e demais funcionários da Universidade Tecnológica Federal do Paraná.

Aos meus colegas de mestrado, pela amizade conquistada e auxílio despendido sempre que necessário.

Para meus amigos, pela convivência, amizade e força para eu enfrentar quaisquer obstáculos.

A minha noiva, por me apoiar sempre, consolar, compreender e me aturar nos momentos mais difíceis, mesmo quando estava com humor péssimo...

”Tudo acontece na hora certa.
Tudo acontece, exatamente quando deve acontecer.”
(Albert Einstein)

”O que sabemos é uma gota, o que ignoramos é um oceano.”
(Isaac Newton)

RESUMO

SCALABRIN, Marlon H.. Mega Busca Harmônica: Algoritmo de Busca Harmônica Baseado em População e Implementado em Unidades de Processamento Gráfico. 132 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná – UTFPR. Curitiba, 2012.

Este trabalho propõe uma modificação da meta-heurística Busca Harmônica (HS) a partir de uma nova abordagem baseada em população, empregando, também, algumas estratégias inspiradas em outras meta-heurísticas. Este novo modelo foi implementado utilizando a arquitetura de programação paralela CUDA em uma GPU. O uso de placas de processamento gráficas (GPU) para processamento de propósito geral está crescendo, e estas têm sido utilizadas por muitos pesquisadores para processamento científico. Seu uso se mostra interessante para meta-heurísticas populacionais, podendo realizar muitas operações simultaneamente. A HS é uma meta-heurística inspirada no objetivo de um músico em buscar uma harmonia perfeita. No modelo proposto incluiu-se uma população de harmonias temporárias que são geradas a cada nova iteração, permitindo a realização simultânea de diversas avaliações de função. Assim aumenta-se o grau de paralelismo da HS, possibilitando maiores ganhos de velocidade com o uso de arquiteturas paralelas. O novo modelo proposto executado em GPU foi denominado *Mega Harmony Search* (MHS). Na implementação em GPU cada passo do algoritmo é tratado individualmente em forma de *kernels* com configurações particulares para cada um. Para demonstrar a eficácia do modelo proposto foram selecionados alguns problemas de *benchmark*, como a otimização de estruturas de proteínas, a otimização de treliças e problemas matemáticos. Através de experimentos fatoriais foi identificado um conjunto de parâmetros padrão, o qual foi utilizado nos outros experimentos. As análises realizadas sobre resultados experimentais mostram que o MHS apresentou solução de qualidade equivalente à HS e ganhos de velocidade, com a sua execução em GPU, superiores a 60x quando comparado a implementação em CPU. Em trabalhos futuros poderão ser estudadas novas modificações ao algoritmo, como a implementação de nichos e estudos de estratégias de interação entre eles.

Palavras-chave: Busca Harmônica, Meta-heurística baseada em População, GPU, CUDA.

ABSTRACT

SCALABRIN, Marlon H.. Mega Harmony Search: Population-Based Harmony Search Algorithm Implemented on Graphic Processing Units. 132 p. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná – UTFPR. Curitiba, 2012.

This work propose a new approach for the metaheuristic Harmonic Search (HS), by using a population of solutiona and other strategies inspired in another metaheuristics. This new model was implemented using a parallel architecture of a graphical processing unity (GPU). The use of GPU for general-purpose processing is growing, specially for scientific processing. Its use is particularly interesting for populational metaheuristics, where multiple operations are executed simultaneously. The HS is a metaheuristic inspired by the way jazz musicians search for a perfect harmony. In the proposed model a population of temporary harmonies was included. Such population was generated at each iteration, enabling simultaneous evaluation of the objective function being optimized, and thus, increasing the level of parallelism of HS. The new approach implemented in GPU was named Mega Harmony Search (MHS), and each step of the algorithm is handled in the form of kernels with particular configurations for each one. To show the efficiency of MHS some benchmark problems were selected for testing, including mathematical optimization problems, protein structure prediction, and truss structure optimization. Factorial experiments were done so as to find the best set of parameters for the MHS. The analyzes carried out on the experimental results show that the solutions provided by MHS have comparable quality to those of the simple Harmony Search. However, by using GPU, MHS achieved a speedup of 60x, compared with the implementation in regular CPU. Future work will focus other improvements in the algorithm, such as the use of niches and species, as well a study of the interactions between them.

Keywords: Harmony Search, Population-based meta-heuristic, GPU, CUDA.

LISTA DE FIGURAS

FIGURA 1	– Comparativo de desempenho de CPUs e GPUs	20
FIGURA 2	– Distribuição de <i>threads</i> e blocos em uma grade computacional de uma GPU.	22
FIGURA 3	– Permissões e escopo de memórias	23
FIGURA 4	– Analogia entre improviso e otimização.	27
FIGURA 5	– Fluxograma da Busca Harmônica.	28
FIGURA 6	– Trastes no braço de um violão.	31
FIGURA 7	– Estrutura de um aminoácido.	32
FIGURA 8	– Representação genérica a estrutura de uma proteína hipotética.	35
FIGURA 9	– Geometria inicial da treliça de 10 barras.	40
FIGURA 10	– Geometria inicial da treliça de 200 barras.	41
FIGURA 11	– Função de Griewank.	43
FIGURA 12	– Função de Rosenbrock.	44
FIGURA 13	– Função de Schaffer.	44
FIGURA 14	– Curva normal.	46
FIGURA 15	– Distribuição de <i>threads</i> (T) e blocos (B) no primeiro <i>kernel</i> do processo de inicialização.	53
FIGURA 16	– Distribuição de <i>threads</i> (T) e blocos (B) no segundo <i>kernel</i> do processo de inicialização.	53
FIGURA 17	– Distribuição de <i>threads</i> (T) e blocos (B) no processo de improviso.	54
FIGURA 18	– Distribuição de <i>threads</i> (T) e blocos (B) no processo de atualização.	55
FIGURA 19	– Torneio Estocástico de tamanho 5.	56
FIGURA 20	– Exemplo do uso de blocos construtivos maiores na HS.	58
FIGURA 21	– Exemplo do uso harmonia base na HS.	59
FIGURA 22	– Histograma da qualidade de solução dos experimentos realizados para o dobramento de proteínas com 21 aminoácidos.	65
FIGURA 23	– Gráfico <i>boxplot</i> dos experimentos selecionados para o problema de otimização de treliças para a instância de 200 barras.	66
FIGURA 24	– Curvas de convergência do problema de dobramento de proteínas, 13 aminoácidos	68
FIGURA 25	– Curvas de convergência do problema de dobramento de proteínas, 21 aminoácidos	68
FIGURA 26	– Curvas de convergência do problema de dobramento de proteínas, 34 aminoácidos	69
FIGURA 27	– Curvas de convergência do problema de otimização estrutural de treliças, 10 barras	69
FIGURA 28	– Curvas de convergência do problema de otimização estrutural de treliças, 200 barras	70
FIGURA 29	– Curvas de convergência da função de Griewank com 30 dimensões	70
FIGURA 30	– Curvas de convergência da função de Griewank com 50 dimensões	71
FIGURA 31	– Curvas de convergência da função de Rosenbrock com 30 dimensões	71
FIGURA 32	– Curvas de convergência da função de Rosenbrock com 50 dimensões	72
FIGURA 33	– Curvas de convergência da função de Schaffer com 30 dimensões	72

FIGURA 34	–	Curvas de convergência da função de Schaffer com 50 dimensões	73
FIGURA 35	–	<i>Speed-ups</i> para diferentes instâncias o dobramento de proteínas.	74
FIGURA 36	–	Curvas de convergência para diferentes valores de <i>TS</i>	75
FIGURA 37	–	Curvas de convergência para diferentes valores de <i>MBL</i>	76
FIGURA 38	–	Curvas de convergência para diferentes valores de <i>UBHR</i>	77
FIGURA 39	–	Curvas de convergência para diferentes valores de <i>EXP</i>	77
FIGURA 40	–	Curvas de convergência para diferentes estratégias de autoadaptação	78

LISTA DE TABELAS

TABELA 1	– Instâncias do problema de dobramento de proteínas AB-2D	36
TABELA 2	– Alguns resultados encontrados na literatura para o problema de dobramento de proteínas AB-2D	37
TABELA 3	– Grupos de elementos para a treliça de 200 barras	42
TABELA 4	– Parâmetros utilizados para os experimentos fatoriais com o problema de dobramento AB-2D na sequência de 21 aminoácidos	64
TABELA 5	– Qualidade de solução dos problemas com o uso das meta-heurística HS e MHS	67
TABELA 6	– Resumo dos tempos de processamento para cada problema	73
TABELA 7	– Qualidade de solução e tempo de processamento para diferentes valores de TS	75
TABELA 8	– Qualidade de solução e tempos de processamento para as estratégias de autoadaptação	79
TABELA 9	– Resultados de 324 experimentos realizados para o problema de otimização de dobramento de proteínas AB-2D para a sequência de Fibonacci de 21 aminoácidos	106
TABELA 10	– Resultados de 24 experimentos realizados para o problema de otimização de estrutura de treliças para a instância de 200 barras	122
TABELA 11	– Resultados de 12 experimentos realizados para o problema de otimização estrutural de treliças para a instância de 10 barras	124
TABELA 12	– Resultados de 12 experimentos realizados para o problema de dobramento de proteínas AB-2D para a sequência de Fibonacci de 13 aminoácidos	125
TABELA 13	– Resultados de 12 experimentos realizados para o problema de dobramento de proteínas AB-2D para a sequência de Fibonacci de 34 aminoácidos	126
TABELA 14	– Resultados de 12 experimentos realizados para a otimização do problema matemático Griewank com 30 dimensões	127
TABELA 15	– Resultados de 12 experimentos realizados para a otimização do problema matemático Griewank com 50 dimensões	128
TABELA 16	– Resultados de 12 experimentos realizados para a otimização do problema matemático Rosenbrock com 30 dimensões	129
TABELA 17	– Resultados de 12 experimentos realizados para a otimização do problema matemático Rosenbrock com 50 dimensões	130
TABELA 18	– Resultados de 12 experimentos realizados para a otimização do problema matemático Schaffer com 30 dimensões	131
TABELA 19	– Resultados de 12 experimentos realizados para a otimização do problema matemático Schaffer com 50 dimensões	132

LISTA DE ALGORITMOS

ALGORITMO 1	– Pseudo-código geral de uma meta-heurística baseada em população. .	26
ALGORITMO 2	– Pseudo-código do algoritmo Busca Harmônica.	29
ALGORITMO 3	– Pseudo-código do algoritmo Busca Harmônica baseada em Geração. .	51
ALGORITMO 4	– Pseudo-código do algoritmo da MHS, incluindo as melhorias propostas.	57
ALGORITMO 5	– Código em linguagem C do incremento dos elementos de um vetor. .	91
ALGORITMO 6	– Código em linguagem CUDA do incremento dos elementos de um vetor.	91
ALGORITMO 7	– Código em linguagem CUDA do uso da memória global pela CPU. .	92
ALGORITMO 8	– Código em linguagem CUDA do uso da memória global pela GPU. .	92
ALGORITMO 9	– Código em linguagem C do fluxo principal de controle da MHS.	93
ALGORITMO 10	– Código em linguagem CUDA da primeira parte do processo de inicialização da memória harmônica.	94
ALGORITMO 11	– Código em linguagem CUDA da segunda parte do processo de inicialização da memória harmônica.	95
ALGORITMO 12	– Código em linguagem CUDA do processo de improviso de uma nova harmonia.	96
ALGORITMO 13	– Código em linguagem CUDA do processo de atualização da memória harmônica.	97

LISTA DE SIGLAS

CPU	<i>Central Processing Unit</i> (Unidade Central de Processamento)
CUDA	<i>Computer Unified Device Architecture</i>
EC	<i>Evolutionary Computation</i> (Computação Evolucionária)
GPGPU	<i>General-Purpose Computing on Graphics Processing Unit</i>
GPU	<i>Graphics Processing Unit</i>
HS	<i>Harmony Search</i> (Busca Harmônica)
HM	<i>Harmony Memory</i> (Memória Harmônica)
ISA	<i>Instruction Set Architecture</i>
MHS	<i>Mega Harmony Search</i> (Mega Busca Harmônica)
OpenCL	<i>Open Computing Language</i>
PBHS	<i>Population-Based Harmony Search</i> (Busca Harmônica Baseada em População)
PDB	<i>Protein Data Bank</i>
PSP	<i>Protein Structure Prediction</i> (Predição de Estrutura de Proteínas)
SDK	<i>Software Developers Kit</i>
SI	<i>Swarm Intelligence</i> (Inteligência de Enxames)
SIMD	<i>Single Instruction, Multiple Data</i>
SIMT	<i>Single-Instruction Multiple-Thread</i>

SUMÁRIO

1	INTRODUÇÃO	15
1.1	MOTIVAÇÃO	15
1.2	OBJETIVOS	17
1.2.1	Objetivo Geral	17
1.2.2	Objetivos Específicos	17
1.3	ORGANIZAÇÃO DA DISSERTAÇÃO	17
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	COMPUTAÇÃO BASEADA EM GPU	19
2.1.1	Diferentes Tipos de Memória da Arquitetura CUDA	23
2.2	COMPUTAÇÃO EVOLUCIONÁRIA	25
2.2.1	Busca Harmônica	26
2.2.2	Método das penalidades	32
2.3	PREDIÇÃO DE ESTRUTURA DE PROTEÍNAS	32
2.3.1	O modelo AB-2D <i>off-lattice</i>	34
2.4	OTIMIZAÇÃO ESTRUTURAL DE TRELIÇAS	37
2.4.1	Treliça Plana de 10 barras	39
2.4.2	Treliça Plana de 200 barras	40
2.5	PROBLEMAS DE OTIMIZAÇÃO DE FUNÇÃO	42
2.6	ANÁLISE ESTATÍSTICA	44
2.6.1	Análise de Variância	44
2.6.2	Teste de normalidade	45
2.6.3	Teste de Scott-Knott	46
2.7	TRABALHOS RELACIONADOS	47
3	METODOLOGIA	49
3.1	DESCRIÇÃO DO TRABALHO	49
3.2	IMPLEMENTAÇÃO DA BUSCA HARMÔNICA	49
3.3	BUSCA HARMÔNICA BASEADA EM GERAÇÕES	50
3.4	MEGA BUSCA HARMÔNICA	52
3.5	ESTRATÉGIAS PROPOSTAS	55
3.5.1	Método de seleção	56
3.5.2	Blocos construtivos maiores	58
3.5.3	Harmonia base	59
3.5.4	Explosão ou Dizimação	59
3.5.5	Autoadaptação de Parâmetros	60
4	RESULTADOS EXPERIMENTAIS	62
4.1	ORGANIZAÇÃO DOS EXPERIMENTOS	62
4.2	IDENTIFICAÇÃO DE PARÂMETROS	63
4.3	ANÁLISE DA QUALIDADE DE SOLUÇÃO	66
4.4	ANÁLISE DE CONVERGÊNCIA	67
4.5	ANÁLISE DE DESEMPENHO	73
4.6	ANÁLISE DAS ESTRATÉGIAS PROPOSTAS	74

4.7 CONSIDERAÇÕES GERAIS	79
5 CONCLUSÕES E TRABALHOS FUTUROS	81
REFERÊNCIAS	83
APÊNDICE A – IMPLEMENTAÇÃO USANDO CUDA	90
APÊNDICE B – ALGORITMOS DO MHS	93
APÊNDICE C – ANÁLISES ESTATÍSTICAS	98
C.1 IDENTIFICAÇÃO DE PARÂMETROS - PRIMEIRA PARTE	98
C.2 IDENTIFICAÇÃO DE PARÂMETROS - SEGUNDA PARTE	100
C.3 ANÁLISE DAS ESTRATÉGIAS PROPOSTAS	101
APÊNDICE D – RESULTADOS DOS EXPERIMENTOS FATORIAIS	105
D.1 DADOS EXPERIMENTAIS PARA AJUSTE DE PARÂMETROS	105
D.1.1 Problema de dobramento de Proteínas, 21 amoniácidos	105
D.1.2 Otimização estrutural de treliças, 200 barras	122
D.2 DADOS EXPERIMENTAIS DAS ANÁLISES DE RESULTADOS	124
D.2.1 Otimização estrutural de treliças, 10 barras	124
D.2.2 Problema de dobramento de Proteínas, 13 amoniácidos	125
D.2.3 Problema de dobramento de Proteínas, 34 amoniácidos	126
D.2.4 Problema matemático Griewank com 30 dimensões	127
D.2.5 Problema matemático Griewank com 50 dimensões	128
D.2.6 Problema matemático Rosenbrock com 30 dimensões	129
D.2.7 Problema matemático Rosenbrock com 50 dimensões	130
D.2.8 Problema matemático Schaffer com 30 dimensões	131
D.2.9 Problema matemático Schaffer com 50 dimensões	132

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Unidades de Processamento Gráfico (*Graphics Processing Units* – GPUs) são dispositivos desenvolvidos originalmente com o objetivo de processamento gráfico de imagens digitais. Todavia, recentemente, as GPUs começaram a ser utilizadas para o desenvolvimento de aplicações computacionais de propósito geral que requerem alto desempenho. Graças a sua ampla capacidade de processamento paralelo e velocidade em operações de ponto flutuante, as GPUs possibilitam um desempenho computacional impressionante e, como consequência, muitas aplicações científicas e de engenharia têm aparecido recentemente. Veja, por exemplo, Komatitsch et al. (2010), Moorkamp et al. (2010), Scanzio et al. (2010), Shams et al. (2010), Sunarso, Tsuji e Chono (2010) e Scalabrin, Parpinelli e Lopes (2010).

Para incentivar o uso de GPUs para computação de propósito geral, algumas plataformas foram desenvolvidas, como a BrookGPU (Universidade de Stanford) (BUCK et al., 2004), a CUDA (*Computer Unified Device Architecture*) (NVIDIA Corporation, 2007) e OpenCL (*Open Computing Language*) (APPLE COMPUTER INC., 2008; KHROS OpenCL WORKING GROUP, 2010). Estas plataformas têm simplificado significativamente a tarefa de programação em GPU. Porém, tirar proveito da capacidade de processamento paralelo de uma GPU é tarefa difícil, visto que existem diferentes possibilidades de implementação e uso de memória, as quais podem afetar de forma bastante expressiva a velocidade do processamento. Nos últimos anos, o uso de GPU para o processamento de problemas resolvidos por meta-heurísticas vem se mostrando bastante popular, principalmente para cálculo simultâneo das funções objetivo em meta-heurísticas populacionais (YU; CHEN; PAN, 2005; BOŽEJKO; SMUTNICKI; UCHROŃSKI, 2009; TAN; ZHOU, 2010).

A cada iteração todas as soluções candidatas devem ser avaliadas. Uma vez que tal avaliação repete o mesmo processo, o uso de processamento paralelo pode ser muito útil para reduzir o tempo de processamento global, proporcionalmente ao tamanho da população a ser avaliada. Em geral, quanto maior a população, maior a redução do tempo de processamento

de uma geração. Por outro lado, quanto maior a população, maior o tempo necessário para transferência entre memórias, sendo este um limitador.

Uma meta-heurística relativamente recente é a Busca Harmônica (*Harmony Search* – HS), introduzida em 2001 por Geem, Kim e Loganathan (2001). Ela tem sido aplicada com sucesso em inúmeros problemas, apresentando bons resultados com um número reduzido de avaliações de função (AYVAZ, 2007; VASEBI; FESANGHARY; BATHAEE, 2007; MAH-DAVI; FESANGHARY; DAMANGIR, 2007; FESANGHARY et al., 2008; GEEM, 2010). Como apresentado em Scalabrin, Parpinelli e Lopes (2010), é possível obter ganhos de velocidade de processamento da HS com o uso de GPU. Porém, este ganho está limitado a problemas com um número elevado de dimensões e problemas complexos com alto nível de paralelismo. Para os casos gerais, o fator decisivo para a redução do tempo de processamento do algoritmo em GPU é a natureza implicitamente paralela da execução de seus passos. Para problemas com número reduzido de variáveis, o uso de GPU não apresentou bons resultados, necessitando de um tempo relativamente grande para encontrar boas soluções. Isto ocorre devido ao grande número de iterações, sendo que apenas uma avaliação é realizada a cada iteração, tornando elevado o custo computacional com muitas transferências entre memórias e de chamadas de *kernel*.

Uma adaptação da HS, incluindo uma população de indivíduos temporários regenerados a cada iteração, mostra-se promissora pela possibilidade da realização de diversas avaliações de função simultâneas, aumentando o nível de paralelismo e possibilitando maiores ganhos em desempenho.

Neste trabalho é proposto um modelo baseado em população para a meta-heurística Busca Harmônica, permitindo, assim, o cálculo paralelo da função de avaliação de diversas harmonias simultaneamente. Desta maneira, pode-se tirar maior proveito do poder de processamento massivamente paralelo de uma GPU.

Dois problemas particularmente interessantes de serem implementados em arquiteturas paralelas são: o problema de dobramento de proteínas, que é de difícil solução, sendo caracterizado pela não-linearidade e fortes restrições (ATKINS; HART, 1999), (BERGER; LEIGHTON, 1998) e (CRESCENZI et al., 1998); e a otimização estrutural de treliças, que possui características de paralelismo em alguns de seus cálculos. Devido à inexistência de métodos exatos para resolver estes problemas, surge a necessidade de técnicas robustas, como as meta-heurísticas evolucionárias. Ao longo de décadas, Computação Evolucionária (*Evolutionary Computation* – CE) e Inteligência de Enxames (*Swarm Intelligence* – SI) têm fornecido uma ampla gama de algoritmos de otimização robustos e flexíveis, capazes de lidar com problemas

complexos de otimização.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver e implementar melhorias à meta-heurística HS, visando um maior nível de paralelismo, mantendo ou melhorando a qualidade das soluções obtidas.

1.2.2 Objetivos Específicos

- Propor melhorias à meta-heurística Busca Harmônica (GEEM; KIM; LOGANATHAN, 2001), incluindo uma população de harmonias temporárias;
- Implementar o novo modelo proposto em arquitetura CUDA, adicionando aperfeiçoamentos inspirados em outras meta-heurísticas populacionais;
- Executar experimentos fatoriais utilizando algumas configurações de parâmetros de controle para a HS original e para o novo modelo, visando identificar as melhores configurações;
- Aplicar o conjunto de parâmetros selecionados para a execução de outros problemas, verificando sua validade;
- Analisar estatisticamente o novo modelo proposto, investigando o desempenho (tempo de processamento) e a qualidade de solução, comparados ao algoritmo original.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

Nesta dissertação o Capítulo 2 apresentam-se alguns conceitos sobre computação baseada em GPU detalhando as características gerais da arquitetura CUDA, bem como seus tipos de memória e modelos de implementação. Na sequência são apresentados conceitos básicos de Computação Evolucionária, em especial a meta-heurística Busca Harmônica. São também apresentados os problemas de *benchmark* que serão utilizados nos experimentos. Por fim, são descritas as análises estatísticas que serão utilizadas na análise dos resultados e apresentados alguns trabalhos relacionados.

O Capítulo 3 descreve a implementação da Busca Harmônica em CPU (*Central Processing Unit*). Apresentando na sequência a Busca Harmônica baseada em população e sua implementação em GPU combinada a diversas estratégias inspiradas em outras meta-heurísticas, as quais também são descritas, sendo esta combinação intitulada Mega Busca Harmônica (*Mega Harmony Search – MHS*).

No Capítulo 4 relatam-se os experimentos realizados e os resultados obtidos. Neste capítulo inicialmente identificou-se um conjunto de parâmetros através de experimentos fatoriais, os quais foram aplicados na resolução de diversos problemas. Os resultados obtidos foram então analisados de acordo com critérios de qualidade de solução, comportamento da curva de convergência e desempenho. Também foi realizada uma análise da influência de cada estratégia, independentemente, sobre a MHS.

E, por fim, o Capítulo 5 apresenta as conclusões do trabalho e algumas propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO BASEADA EM GPU

Nas últimas três décadas, os processadores de computadores *desktop* (*Central Processing Unity* – CPU) apresentaram melhoras significativas. De acordo com a Lei de Moore (MOORE, 1965), o poder de processamento, ou a eficiência dos computadores, têm duplicado a cada 18 meses, conforme confirmado recentemente pela pesquisa de Koomey et al. (2011). Todavia, o desempenho das GPUs têm melhorado a uma taxa extraordinária desde que elas apareceram, mais rápido do que estimado por esta lei (MOHANTY, 2009). Este fato tem chamado a atenção de pesquisadores e desenvolvedores para o uso de GPUs para aplicações de propósito geral, e não apenas processamento gráfico. Como consequência, as GPUs estão se tornando a tecnologia de maior poder computacional para computação científica e de engenharia (OWENS et al., 2005).

A partir de novembro de 2006, com a introdução da arquitetura Tesla – a primeira GPU da Nvidia destinada exclusivamente à computação de propósito geral em unidades de processamento gráfico (*General-Purpose Computing on Graphics Processing Unit* – GPGPU) (NVIDIA CUDA Team, 2009), as GPUs Nvidia passaram a estabelecer um poder computacional significativamente superior às CPUs, conforme mostra a Figura 1, apresentando ganhos de velocidade cada vez maiores, com a expectativa de superar 12 TeraFlops com uma única placa em 2014, mantendo o consumo de energia (NVIDIA CUDA Team, 2008b).

Juntamente com esta nova arquitetura, foi introduzido o conceito de *Single-Instruction Multiple-Thread* (SIMT). Essa arquitetura cria, gerencia, agenda e executa *threads*¹ em grupos, paralelamente, sem *overhead*² de agendamento (NVIDIA CUDA Team, 2009). O fator preponderante para o enorme ganho de desempenho propiciado pela GPU se dá por ela ser dedicada ao processamento paralelo de dados, não gastando poder computacional guardando informações

¹*Thread* é um fluxo único de controle sequencial dentro de um programa

²*Overhead* é um termo comum que significa sobrecarga ou excesso, podendo ser considerado um "tempo de espera"

automaticamente em *cache*³, possuindo, também, um fluxo de execução simples, com execução linear e independente de suas *threads*.

Recentemente, a Corporação NVIDIA desenvolveu a arquitetura CUDA, uma plataforma de desenvolvimento de aplicações de computação paralela diretamente para placas GPU. Com este recurso é possível usar o poder do processamento paralelo através do uso de uma simples extensão da linguagem de programação C (GARLAND et al., 2008). CUDA foi apresentado ao público no início de 2007 como uma SDK (*Software Developers Kit*) com compiladores para sistemas operacionais Windows e Linux (NVIDIA Corporation, 2007).

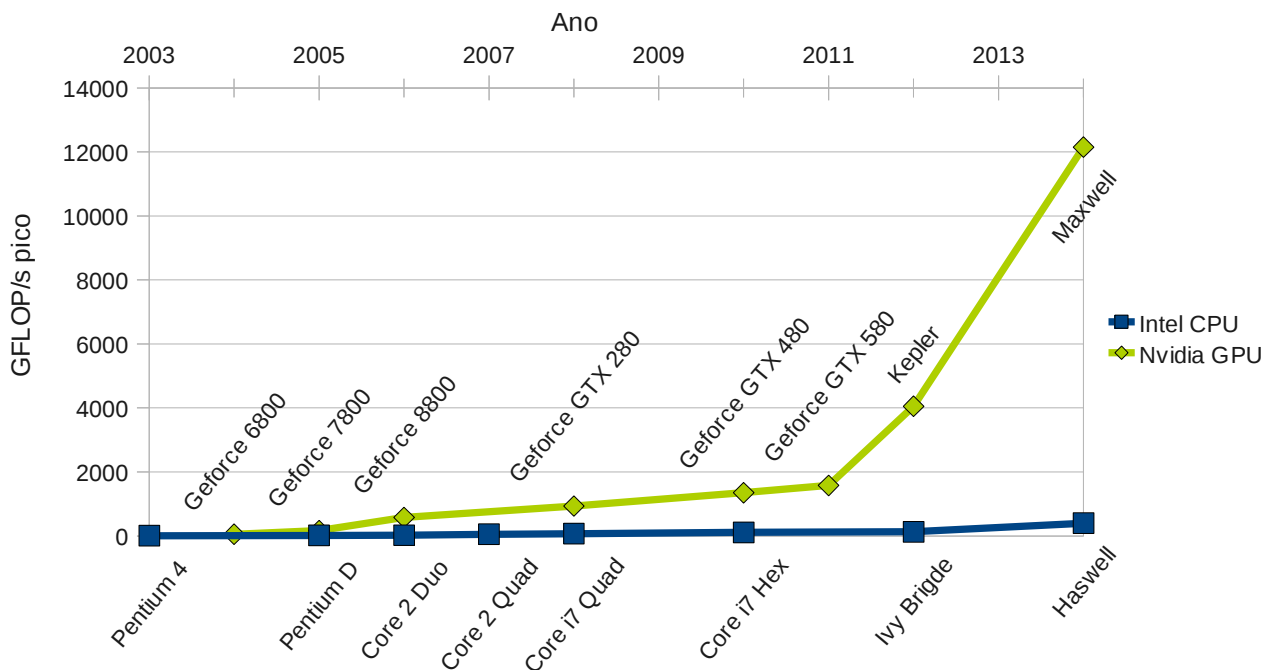


Figura 1 – Comparativo de desempenho de CPUs e GPUs
Fonte: Baseado em NVIDIA CUDA Team (2008b, 2009).

Um programa desenvolvido utilizando CUDA pode tirar vantagem do uso dos multiprocessadores presentes na placa GPU, assim permitindo ganhos de velocidade (*speed-ups*) superiores ao uso de CPUs *multicore* (NVIDIA CUDA Team, 2010). Este ganho de velocidade se dá pela possibilidade da execução paralela de um número muito maior de processos (*threads*) e uma nova forma de organização da arquitetura.

A arquitetura CUDA é composta, principalmente, pelo conjunto de instruções CUDA ISA (*Instruction Set Architecture*) e pelo *hardware* de computação paralela (multiprocessadores) da GPU. CUDA tem sido melhorada e adaptada ao longo do tempo, mantendo sempre a compatibilidade com aplicações previamente desenvolvidas, sendo que atualmente se encontra na versão 4.0.

³Cache é uma memória mais rápida que funciona como interface de acesso a uma memória mais lenta, armazenando dados para acessos recorrentes

O ciclo de execução de uma aplicação CUDA é alternada entre execuções na CPU e na GPU, sendo que a execução de uma tarefa na GPU é realizada através de chamadas de funções denominadas *kernels* que, por sua vez, inicializam várias *threads* idênticas que executam, preferencialmente, a mesma instrução em paralelo na GPU. Cada *thread* é responsável pelo processamento de uma parte de um grande conjunto de dados (NVIDIA CUDA Team, 2009). Este tipo de processamento paralelo é conhecido como *Single Instruction, Multiple Data* – SIMD (PARHAMI, 2002).

A inicialização do *kernel* é feita através de um aplicativo de controle executado pela CPU, sendo configurado conforme a necessidade. Esta configuração deve ser sempre especificada, distribuindo as *threads* na GPU em dois níveis diferentes, de tal maneira que elas são agrupadas em blocos que, por sua vez, são parte de uma grade computacional (*grid*). Isto é feito através de uma nova sintaxe, na qual se utiliza entre o nome do *kernel* e os argumentos do mesmo, um vetor bidimensional onde constam as dimensões do *grid* e do bloco, respectivamente, delimitados pelos prefixo <<< e posfixo >>>, como, por exemplo:

```
nome_do_kernel<<< gridDim, blockDim >>>(arg1, arg2).
```

As funções executadas na GPU são definidas com os qualificadores `__global__` e `__device__`. O primeiro qualificador define as funções do tipo *kernel* que podem ser acessadas pelo fluxo de controle em CPU. O segundo, define as funções de sub-rotinas GPU, acessíveis apenas por outras funções em GPU.

As funções `__global__` são assíncronas, ou seja, a execução na CPU continua mesmo que não tenha terminado a execução em GPU, por este motivo retornando `void`. A definição de um *kernel* é realizada como o seguinte modelo:

```
__global__ void nome_do_kernel(tipo arg1, tipo arg2)
```

As funções executadas em GPU possuem algumas restrições: não possuem suporte à recursão e não podem ter número variável de argumentos. Os argumentos de cada função com qualificador `__global__` são transferidos às *threads* via memória compartilhada, estando limitados a 256 *bytes*, o que permitiria, por exemplo, transferir 64 argumentos do tipo `float`.

A plataforma CUDA oferece uma forma que permite calcular as posições de memória em que os dados a serem manipulados estão disponíveis, baseada em índices e dimensões das *threads*, dos blocos e do *grid*. Cada *thread*, bem como cada bloco, possui um índice único definido pelas variáveis `threadIdx` e `blockIdx`, respectivamente. Os valores das dimensões de um bloco são obtidos através da variável `blockDim`, e os valores das dimensões do *grid*, através da variável `gridDim`. Desta forma, o número total de *threads* tem uma relação direta

com o tamanho dos dados a serem processados e a utilização do valor destas variáveis fornece o índice para o acesso às posições corretas de memória.

A distribuição de blocos em um *grid* pode ser feita em duas dimensões, e a distribuição das *threads* em um bloco pode ser feita em três dimensões. Isto permite organizar os elementos de processamento multidimensionalmente, conforme a distribuição dimensional dos dados que serão processados, organizando, também, para o uso adequado das memórias compartilhadas (ver seção 2.1.1). A Figura 2 mostra um exemplo da distribuição espacial de uma *grid*, detalhando seus blocos e *threads* de forma bi-dimensional, bem como o acesso da CPU (hospedeiro – *host*) ao dispositivo (*device*) GPU.

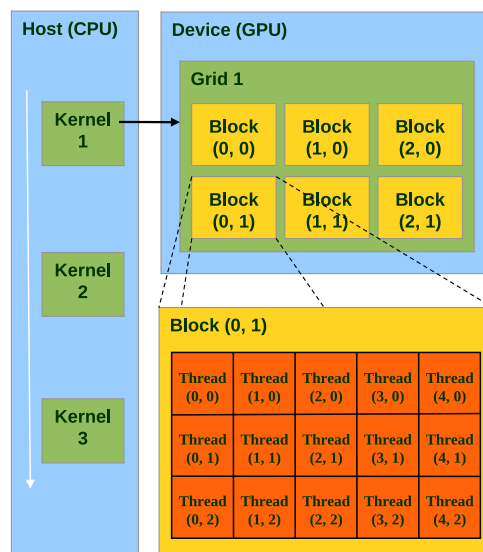


Figura 2 – Distribuição de *threads* e blocos em uma grade computacional de uma GPU.
Fonte: Adaptado de Kirk e Hwu (2009a).

As *threads* de um mesmo bloco podem se comunicar entre si usando os espaços específicos de memória compartilhada, sendo necessária a sincronização em alguns momentos para garantir que os dados corretos serão acessados nesta memória. Desta forma, a sincronização das *threads* de um bloco, através do comando `__syncthreads()`, permite criar um fluxo de controle comum para elas. Isto significa que apenas quando todas as *threads* tenham atingido o mesmo ponto elas poderão prosseguir a execução.

As GPUs atuais são construídas de muitos multiprocessadores com vários núcleos, chamados *Streaming Multiprocessors* (SMs). Os SMs possuem recursos físicos e de memória compartilhada limitados, sendo possível alocar no máximo 768 *threads* em cada SM, independente do número de blocos, tendo o número de *threads* por bloco limitado a 512 (NVIDIA CUDA Team, 2010). Uma vez que cada bloco é totalmente alocado em um único SM, seu tamanho deve ser tal a maximizar o uso de recursos computacionais de um SM (KIRK; HWU, 2008).

Cada SM cria, gerencia e executa as *threads* em grupos de 32, chamados *warps*. Se um bloco tem um tamanho maior do que 32, este se divide em *warps* que são executados sequencialmente. Apenas as *threads* pertencentes a cada *warp* são executadas em paralelo. Embora todas as *threads* de um *warp* iniciem ao mesmo tempo, elas são executadas de forma independente. Apesar disto, a execução de um *warp* é mais eficiente quando as suas 32 *threads* executam simultaneamente a mesma instrução, tendo o mesmo fluxo de execução.

Instruções de desvio condicional operando sobre dados diferentes podem levar a divergências no caminho de execução do bloco, ou seja, os fluxos de execução diferentes para *threads* do mesmo bloco. Em tal situação as *threads* são executadas separadamente e sequencialmente pelo *warp*, diminuindo, assim, de forma significativa o desempenho que pode ser atingido pelo sistema (NVIDIA CUDA Team, 2010).

2.1.1 Diferentes Tipos de Memória da Arquitetura CUDA

Na arquitetura CUDA, há seis tipos diferentes de memória, representadas na Figura 3, cada qual com seus tempo de acesso, permissões, escopo e duração específicos (NVIDIA CUDA Team, 2008a, 2010; KIRK; HWU, 2008):

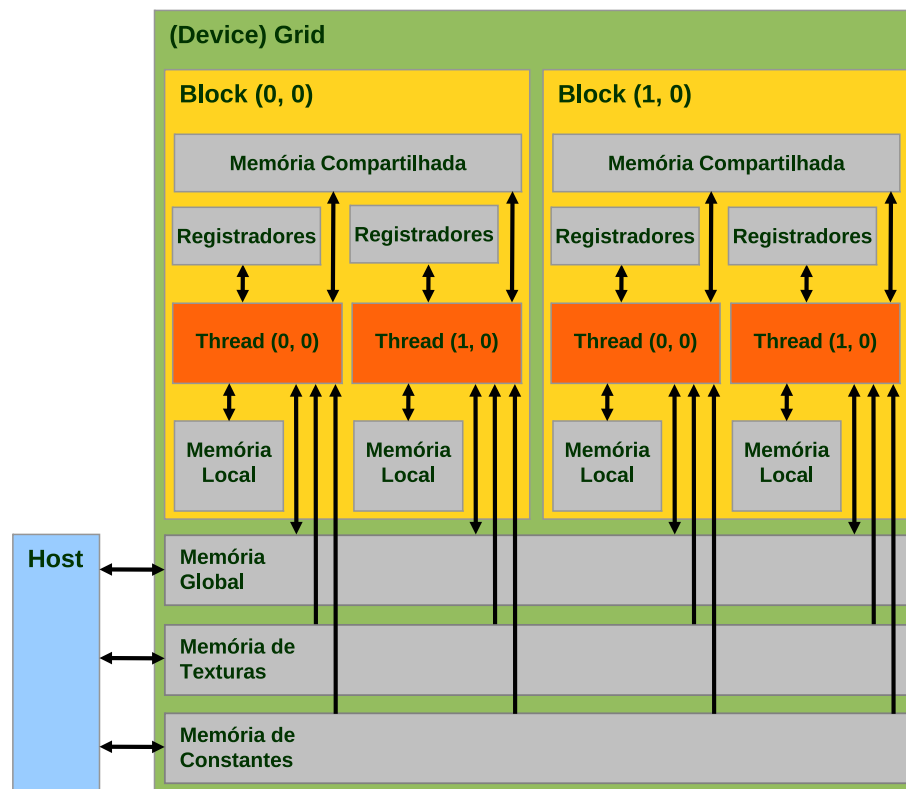


Figura 3 – Permissões e escopo de memórias
 Fonte: Adaptado de Kirk e Hwu (2009b).

1. **Memória de Registradores:** é uma memória de acesso rápido em que tipos primitivos de dados são armazenados (i.e., `int`, `float`, `char`). Registradores têm permissões de leitura e de escrita, o escopo é restrito a cada *thread* e o tempo de vida é equivalente ao de sua *thread*. Não há a necessidade de identificação do propósito destas variáveis. Sua simples declaração dentro da *thread* reserva o espaço de memória dos registradores.
2. **Memória Local:** é a memória endereçável em que as variáveis são armazenadas, tais como ponteiros para valores na memória global e vetores. Tipos de dados primitivos são alocados na memória local quando a memória de registradores está cheia. Esta memória tem as mesmas permissões e escopo da memória de registradores e também é acessada diretamente de dentro de cada *thread*, não havendo necessidade de identificação de seu uso.
3. **Memória Compartilhada:** área de memória que permite a troca de informações entre *threads* que estão executando no mesmo bloco. Esta memória tem acesso rápido e possui permissões de leitura e de escrita. O tempo de vida é o mesmo do seu bloco. O uso da memória compartilhada é definido no momento da sua declaração incluindo-se a palavra-chave `__shared__`, permitindo, assim, que suas informações possam ser acessadas por qualquer *thread* pertencente ao bloco.
4. **Memória Global:** área da memória que possui permissões de leitura e de escrita, escopo global e tem um tempo de acesso lento. Os valores armazenados nesta região de memória são independentes do tempo de execução dos *kernels* e permanecem armazenados durante todo o tempo de execução da aplicação, ou até que o espaço de memória seja liberado. Isto significa que ela pode armazenar informações a serem compartilhadas por diferentes *kernels* ou execuções diferentes do mesmo *kernel*. Esta memória também é usada para compartilhar dados entre o *host* (CPU) e o *device* (GPU) e pode ser alocada ou liberada a qualquer momento durante a execução do programa. Existem dois modos de fazer o uso desta área de memória: utilizando-se a palavra-chave `__device__` na declaração de uma variável global; ou realizando-se a alocação da memória dinamicamente.
5. **Memória de Constantes:** esta área de memória possui acesso lento, com permissões e escopo semelhantes ao da memória global. Uma vez que os valores são armazenados na memória de constantes não é permitida sua alteração. Nas placas GPU atuais, o espaço de memória de constantes é limitado a 64 Kbytes.
6. **Memória de Textura:** esta área de memória é semelhante à memória de constantes, exceto por ter um mecanismo de *cache* automático.

No desenvolvimento de aplicações com CUDA existem muitas possibilidades diferentes de uso dos recursos da GPU (blocos, *threads* e memórias). Dependendo de como estes recursos são utilizados, desempenhos diferentes podem ser obtidos na execução de um *kernel* (KIRK; HWU, 2008). Assim, a forma como o paralelismo é implementado, ajustando o tamanho dos blocos e *threads*, afetam diretamente o desempenho da execução do *kernel*, também, permitindo ou não o uso da memória compartilhada.

Outras características importantes no desenvolvimento CUDA, que afetam diretamente o desempenho, são o número de chamadas de *kernel* e o uso das memórias. A primeira característica aumenta a sobrecarga de comunicação entre CPU e GPU proporcionalmente ao número de chamadas de *kernel*. A segunda diz respeito à utilização de memórias mais rápidas, como memórias locais e compartilhada, sempre que possível, enfatizando o acesso ordenado e coalescido⁴ (*coalesced*) dos endereços da memória global, também evitando conflitos. Tais características são tratadas na seção 3.4

O Apêndice A apresenta alguns exemplos de implementação CUDA, como a conversão de um método C para um *kernel* CUDA e uso das memórias global e compartilhada.

2.2 COMPUTAÇÃO EVOLUCIONÁRIA

Computação Evolucionária (CE) é um ramo da Inteligência Computacional (IC) que utiliza técnicas de otimização inspiradas principalmente na evolução natural dos seres vivos, baseada em Darwin (1859). Nas décadas de 60 e 70 surgiram as primeiras técnicas de CE, como a Programação Evolucionária (FOGEL, 1964), os Algoritmos Genéticos (HOLLAND, 1975) e as Estratégias Evolucionárias (RECHENBERG, 1973). No período que se seguiu houve maior desenvolvimento destas técnicas e o surgimento de novas técnicas, como, por exemplo, Otimização por Enxame de Partículas (KENNEDY; EBERHART, 1995). Na década de 90 surgiu o conceito que unificou essas técnicas em um único conceito, a Computação Evolucionária.

As técnicas apresentadas pela CE são métodos computacionais genéricos que buscam resolver problemas de otimização de forma iterativa com o objetivo de melhorar soluções candidatas no que diz respeito a uma dada medida de qualidade (*fitness*), para tentar encontrar a melhor solução possível (ótimo global). A seguir, serão descritas brevemente algumas técnicas de computação evolucionária.

As meta-heurísticas, como são chamadas as técnicas inspiradas em comportamentos,

⁴Acesso coalescido refere-se ao acesso contínuo e alinhado da memória global, sendo a transferência entre memórias realizada em uma única transação.

não usam o gradiente do problema a ser otimizado, não exigindo que o problema a ser solucionado seja contínuo nem diferenciável. Portanto, podem ser utilizadas em problemas que possuem um espaço de busca não convexo e não linear (YILMAZ; WEBER, 2011).

O Algoritmo 1 mostra um pseudo-código geral de uma meta-heurística baseada em população. O laço principal (entre as linhas 3–7) representa o laço geracional no qual a cada iteração é gerada uma nova população a ser avaliada. A linha 4 define o mecanismo ou critério de seleção da melhor solução, através, por exemplo, da sobrevivência do melhor, como em CE, ou simplesmente o descarte da pior solução. Duas características importantes de uma meta-heurística baseada em população, e também de qualquer meta-heurística em geral, são os processos de intensificação e diversificação. Na linha 5 do algoritmo, intensificação (*exploitation*), entende-se pelo processo de busca local e mais intensiva em torno das melhores soluções, por exemplo, através do operador de *crossover* no Algoritmo Genético (AG), enquanto que a diversificação (*exploration*) permite ao algoritmo explorar o espaço de busca mais globalmente, por exemplo, através do operador de mutação no AG.

Algoritmo 1 Pseudo-código geral de uma meta-heurística baseada em população.

- 1: *Inicializa* a população com soluções candidatas aleatórias;
 - 2: *Avalia* cada solução candidata;
 - 3: **Enquanto** critério de convergência não é satisfeito **Faça**
 - 4: Realiza o processo de *seleção*;
 - 5: Aplica os processos de *intensificação e diversificação*;
 - 6: *Avalia* o novo *pool* de soluções candidatas;
 - 7: **Fim Enquanto**
-

Além das técnicas de computação evolucionária bio-inspiradas existem meta-heurísticas inspiradas em conceitos artificiais, como é o caso da Busca Harmônica.

2.2.1 Busca Harmônica

A meta-heurística Busca Harmônica (*Harmony Search* – HS) é inspirada em conhecimentos musicais, sendo uma analogia a experimentos de músicos de Jazz, que buscam combinações que são esteticamente agradáveis através do processo de improviso e memorização, desde um determinado ponto de vista (função objetivo) (GEEM; KIM; LOGANATHAN, 2001; GEEM, 2009).

A analogia do algoritmo é feita com o comparativo de um trio de Jazz, como apresentado na Figura 4, composto por um violão, um contrabaixo e um saxofone, onde: cada instrumento representa uma variável de decisão, as notas que podem ser tocadas pelos instru-

mentos representam o intervalo de valores de cada variável, a combinação das notas em certo momento (harmonia) representa o vetor solução em determinado instante e a apreciação do público representa a função objetivo.

Por exemplo, o saxofone tem a possibilidade de tocar as notas {Do, Ré, Mi}; o contrabaixo, {Mi, Fá, Sol}; e o violão, {Sol, Lá, Si}. A combinação das notas dos instrumentos formará harmonias que são esteticamente agradáveis ou não ao público.

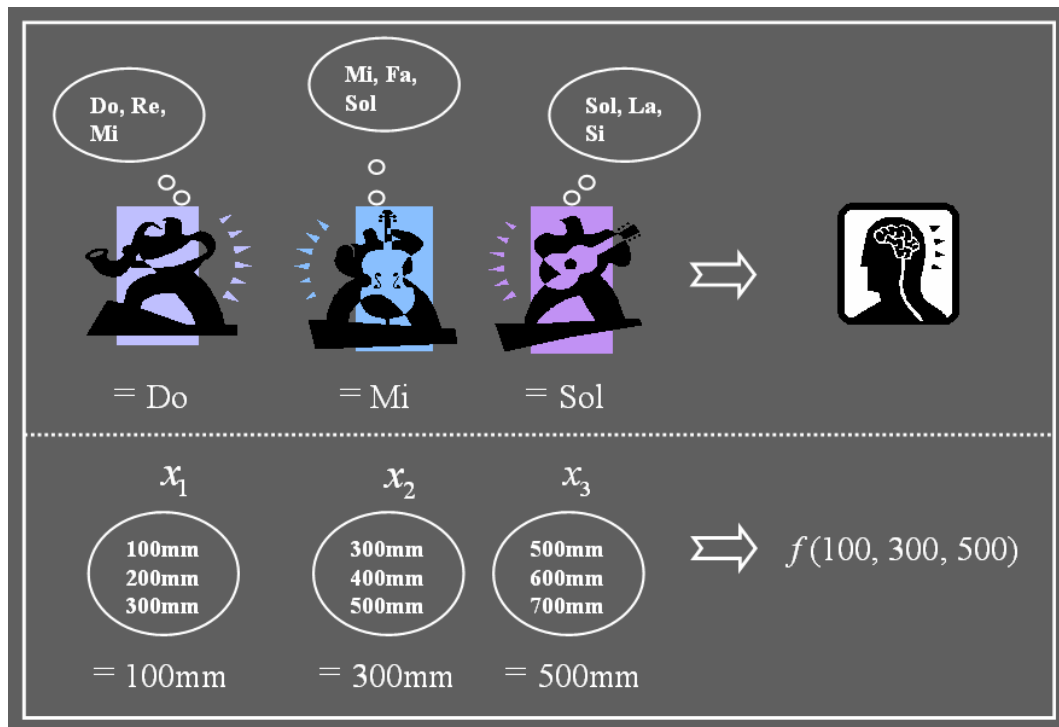


Figura 4 – Analogia entre improviso e otimização.
Fonte: Geem (2010).

A Figura 5, apresenta um fluxograma do algoritmo de Busca Harmônica, partindo da inicialização da Memória Harmônica (*Harmony Memory* – HM) de tamanho *HMS* (*Harmony Memory Size*), na qual cada posição da memória é ocupada por uma harmonia. Neste contexto a Harmonia é um vetor de tamanho *N* (número de músicos) que representa uma possível solução para o problema, no processo de otimização.

A Busca Harmônica não usa o gradiente para realizar a busca, mas, sim, uma busca estocástica aleatória baseada em dois parâmetros principais *HMCR* e *PAR* (GEEM; KIM; LOGANATHAN, 2001).

A cada iteração do algoritmo é improvisada uma nova harmonia a partir das harmonias presentes na memória harmônica, como apresentado no Algoritmo 2 entre as linhas 9 e 16. Se a nova harmonia gerada for melhor que a pior harmonia da memória harmônica, esta é substituída pela nova.

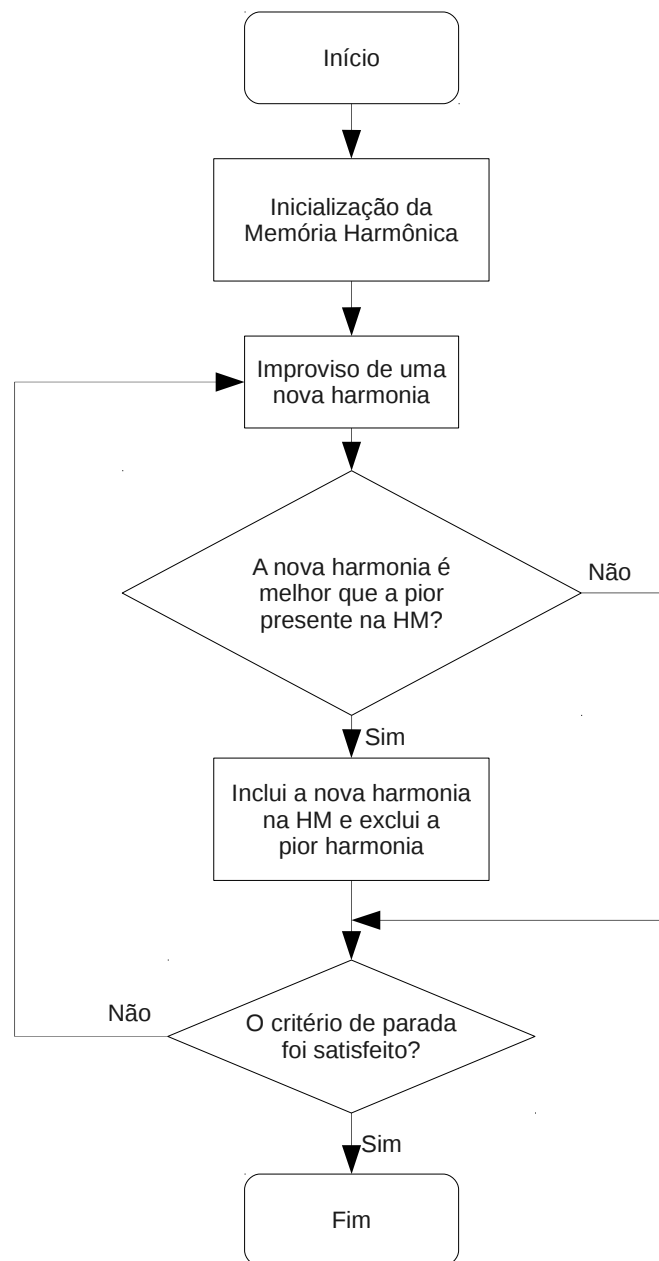


Figura 5 – Fluxograma da Busca Harmônica.
Fonte: Autoria própria.

Os passos de improviso e atualização da memória harmônica são repetidos até que um critério de parada ou o número máximo de improvisos (*MI*) seja alcançado.

Como apresentado por Geem, Kim e Loganathan (2001) e Mahdavi, Fesanghary e Damangir (2007), o algoritmo HS pode ser descrito por cinco passos principais, detalhados a seguir. Em cada passo será apontada a linha referente no pseudo-código apresentado no Algoritmo 2⁵.

⁵Outras informações sobre a HS e aplicações podem ser encontradas em seu repositório (*HS Repository*: <http://www.hydroteq.com>)

Algoritmo 2 Pseudo-código do algoritmo Busca Harmônica.

```

1: Parâmetros: HMS, HMCR, PAR, MI, FW
2: Início
3: Função Objetivo  $f(\vec{x}), \vec{x} = [x_1, x_2, \dots, x_N]$ 
4: Inicialização da Memória Harmônica  $x^i, i = 1, 2, \dots, HMS$ 
5: Avalia cada Harmonia na HM:  $f(x^i)$ 
6: ciclo  $\leftarrow 1$ 
7: Enquanto ciclo < MI Faça
8:   Para j  $\leftarrow 1$  até N Faça
9:     Se  $gera\_aleatorio() \leq HMCR$  Então {Taxa de Consideração da Memória}
10:     $x_j \leftarrow x_j^i$ , com  $i \in [1, HMS]$  {escolhido aleatoriamente}
11:    Se  $gera\_aleatorio() \leq PAR$  Então {Taxa de Ajuste Fino}
12:     $x_j \leftarrow x_j' \pm u(-1, 1) \times FW$ 
13:    Fim Se
14:    Senão {Seleção Aleatória}
15:    Gera  $x_j'$  aleatoriamente
16:    Fim Se
17:  Fim Para
18:  Avalia a nova Harmonia gerada:  $f(x')$ 
19:  Se  $f(x')$  é melhor que a pior Harmonia na HM Então
20:    Atualiza Memória Harmônica
21:  Fim Se
22:  ciclo  $\leftarrow$  ciclo + 1
23:  Verifica Critério de parada
24: Fim Enquanto
25: Exibição dos resultados
26: Fim

```

1. **Inicialização do Problema e Parâmetros do Algoritmo:** No primeiro passo, como em todo problema de otimização, o problema é definido como uma função objetivo a ser otimizada (linha 3), a qual pode ou não possuir um conjunto de restrições implementadas como penalidades (ver seção 2.2.2). Originalmente, a Busca Harmônica foi desenvolvida para a resolução de problemas de minimização (GEEM; KIM; LOGANATHAN, 2001), assim, quanto menor o valor da função objetivo, melhor a qualidade da solução.

Neste passo também são definidos os parâmetros do algoritmo. Os quatro principais parâmetros são o tamanho da memória harmônica (*Harmony Memory Size – HMS*), a taxa de escolha de um valor da memória (*Harmony Memory Considering Rate – HMCR*), a taxa de ajustes dos valores (*Pitch Adjusting Rate – PAR*) e o número máximo de improvisos (*Maximum Improvisation – MI*).

2. **Inicialização da Memória Harmônica:** No segundo passo, a Memória Harmônica é inicializada (linha 4) com um número de harmonias geradas aleatoriamente baseada em uma distribuição uniforme. A Memória Harmônica é o vetor que contém *HMS* harmo-

nias, como apresentado na Equação 1, onde são armazenadas as melhores harmonias encontradas durante a execução do algoritmo.

$$HM = \begin{bmatrix} \{ x_1^1 & x_2^1 & \cdots & x_N^1 \} \\ \{ x_1^2 & x_2^2 & \cdots & x_N^2 \} \\ \vdots & \vdots & \ddots & \vdots \\ \{ x_1^{HMS} & x_2^{HMS} & \cdots & x_N^{HMS} \} \end{bmatrix} \Rightarrow \begin{matrix} f(\vec{x}^1) \\ f(\vec{x}^2) \\ \vdots \\ f(\vec{x}^{HMS}) \end{matrix} \quad (1)$$

3. Improviso de uma nova Harmonia: No terceiro passo é improvisado um novo vetor harmonia, $x' = (x'_1, x'_2, \dots, x'_N)$, baseado nas harmonias existentes na HM (linhas 8 a 17), sendo a nova harmonia uma combinação de várias outras. Para cada variável da nova harmonia seleciona-se arbitrariamente uma harmonia da HM, através de um valor aleatório de uma distribuição uniforme. Verifica-se então a probabilidade deste valor ser ou não utilizado (*HMCR* – linhas 9 e 10; 14 e 15). Desta forma, conforme a Equação 2, se for utilizado o valor de outra harmonia, a posição da nova harmonia recebe a posição correspondente da harmonia selecionada. Se não for utilizado o valor de outra harmonia, um valor aleatório dentro do intervalo de valores permitidos (representado por X_i) é atribuído.

$$x'_i = \begin{cases} x'_i \in \{x_i^1, x_i^2, \dots, x_i^{HMS}\} & \text{com probabilidade HMCR (linha 9)} \\ x'_i \in X_i & \text{com probabilidade } 1 - \text{HMCR (linha 14)} \end{cases} \quad (2)$$

Todo valor recuperado da HM para a composição da nova harmonia pode sofrer pequenos ajustes conforme $x'_i = x'_i + u(-1, 1) \times FW$, onde *FW* (*Fret Width*) é o valor máximo do ajuste e $u(-1, 1)$ sendo um valor aleatório de uma distribuição uniforme entre -1 e 1 (linhas 11 e 12). O controle da ocorrência deste evento é feito pelo parâmetro *PAR*, sendo que a probabilidade de uma variável sofrer o processo de ajuste fino é de $PAR \times HMCR$. O nome *Fret Width*, que significa largura de traste, se refere à distância entre trastes vizinhos de instrumentos de corda como um violão (Figura 6), representando notas musicais com diferenças de semitons.

A combinação dos parâmetros *HMCR* e *PAR* do algoritmo HS é responsável por estabelecer um balanço entre a busca global e a busca local no espaço de busca.

4. Atualização da Memória Harmônica: No quarto passo, a nova harmonia improvisada é avaliada pela função objetivo. Caso a nova harmonia seja melhor do que a pior harmonia da HM, em termos de qualidade de solução (*fitness*), a nova harmonia é incluída na HM, enquanto que a pior harmonia é eliminada (linhas 19 a 21).



**Figura 6 – Trastes no braço de um violão.
Fonte: Autoria própria.**

A versão original da HS apresentado por Geem, Kim e Loganathan (2001) representa a HM como um vetor ordenado pela qualidade de solução, todavia como a seleção dos indivíduos depende de um processo aleatório, manter ou não a memória harmônica ordenada não interfere no processo de otimização. Estando a memória harmônica ordenada, a inserção do novo indivíduo pode ser realizada por métodos como *insertion sort*, no qual o novo indivíduo é inserido na sua posição já ordenado; as posições subsequentes da memória são deslocadas, descartando-se a pior solução. Assim, o melhor indivíduo permanece na primeira posição e o pior na última, não havendo a necessidade de um processo de ordenação. Já quando a memória independe de ordenação ocorre a simples substituição do pior indivíduo pelo novo, como apresentado por Ayvaz (2007) e Vasebi, Fesanghary e Bathaee (2007).

5. **Verificação do critério de parada:** No quinto passo, ao término de cada iteração, é verificado se a melhor harmonia satisfaz o critério de parada (linha 23), sendo, normalmente, o número máximo de improvisos *MI*. Enquanto o critério de parada não for satisfeito, o algoritmo continua a execução, retornando para o segundo passo.

HS é amplamente empregada em processos de otimização em diferentes áreas, tendo obtido soluções competitivas com outros métodos semelhantes na solução de inúmeros problemas de *benchmark* (GEEM; KIM; LOGANATHAN, 2001; LEE; GEEM, 2004; GEEM, 2007, 2009, 2010; SAKA; M.P., 2007; FESANGHARY et al., 2008).

2.2.2 Método das penalidades

O método de penalidade (também conhecido como minimização irrestrita pelo ponto exterior) é o de implementação mais simples para tratamento de restrições em problemas de otimização, sendo, assim, muito utilizado para tratamento de restrições em meta-heurísticas (como é o caso da HS). Sobre o problema restrito, é aplicada uma função de transformação, envolvendo uma função de penalidade ϕ^k e um parâmetro r^k sempre positivo, conforme a Equação 3 (OLIVEIRA, 1989). O objetivo da função de penalidade é aplicar um alto custo à violação das restrições do problema.

$$\phi^k(\vec{x}, r^k) = f(\vec{x}) + r^k P(\vec{x}) \quad \text{com } r^k > 0 \quad (3)$$

Onde, \vec{x} é o vetor de entrada, $P(x)$ representa o termo de penalidade, sendo sempre positivo, apresentando valor maior do que zero sempre que houver a violação de alguma restrição. O parâmetro r^k é o peso multiplicador da penalidade, de forma que quanto maior a restrição, maior a penalidade aplicada. Sendo assim, para que a solução seja válida a penalidade deve tender a zero ($P(x) \rightarrow 0$), onde temos $\phi^k(\vec{x}, r^k) \rightarrow f(x)$.

Dentre as várias formas de construção do termo de penalidade, a função de penalidade mais comum é a proporcional ao quadrado da violação (HAFTKA; GÜRDAL, 1992).

2.3 PREDIÇÃO DE ESTRUTURA DE PROTEÍNAS

Proteínas são as estruturas básicas de todos os seres vivos, desempenhando diversas funções vitais no organismo (HUNTER, 1993). Elas são longas cadeias compostas por um grande número aminoácidos que estão ligados entre si por meio de ligações peptídicas. Cada aminoácido é caracterizado por um átomo de carbono central (também chamado de carbono alfa – $C\alpha$) ao qual estão ligados um átomo de hidrogênio, um grupo carboxila (COOH), um grupo amino ($-NH_2$) e um quarto composto conhecido como cadeia lateral que define uma função distinta para cada aminoácido (BRANDEN; TOOZE, 1999), conforme mostra a Figura 7.

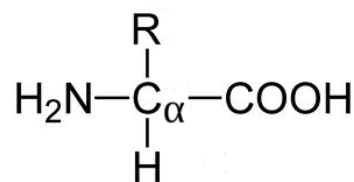


Figura 7 – Estrutura de um aminoácido.
Fonte: Autoria própria.

Existem na natureza uma grande variedade de aminoácidos, mas apenas 20 deles são proteínogênicos. Eles diferem em sua hidrofobicidade e podem ser classificados em duas classes, de acordo com sua afinidade com a água: hidrofílicos (ou Polares) e hidrofóbicos (ou Apolares) (LI; TANG; WINGREEN, 1997). De acordo com este comportamento, pode-se concluir que a polaridade da cadeia lateral governa o processo de formação de estruturas de proteínas (LODISH et al., 2000).

Uma proteína é formada, normalmente, por um grande número de ligações peptídicas. Desta forma, ela é conhecida como uma cadeia polipeptídica (AMABIS; MARTHO, 1990). O dobramento de proteínas é o processo pelo qual estas cadeias polipeptídicas são transformadas em estruturas compactas que desempenham funções biológicas. Estas funções incluem o controle e regulação de processos bioquímicos essenciais para os organismos vivos. Para que a proteína execute sua função fisiológica ela deve encontrar-se em sua conformação nativa, i.e., sua estrutura tridimensional única obtida em condição normais de seu meio natural (TANG, 2000; PEDERSEN, 2000). Espontaneamente, as proteínas buscam o mais alto grau de organização e eficiência na utilização de energia (HENEINE, 1984). Neste estado, a proteína se encontra ativa, exercendo sua atividade biológica, tais como: catálise, transporte, defesa, etc.

Falhas no dobramento para a obtenção da conformação tridimensional, geralmente, geram proteínas com propriedades diferentes das de sua conformação nativa, que simplesmente tornam-se inativas. No pior dos casos, tais proteínas mal-formadas (dobradas incorretamente) podem ser prejudiciais ao organismo. Por exemplo, acredita-se que várias doenças, tais como mal de Alzheimer, fibrose cística e alguns tipos de câncer resultam do acúmulo de proteínas mal-formadas (DOBSON, 1999; THOMASSON, 2001).

O Paradoxo de Levinthal (LEVINTHAL, 1968) é um fator importante a ser analisado no estudo do dobramento, o qual diz que uma proteína dobra-se em um espaço de tempo relativamente curto, no máximo poucos segundos. Como o espaço de busca de soluções possíveis é imenso, é inviável a busca da conformação ideal utilizando um método que avalie todas as possíveis conformações para uma determinada sequência.

Sabe-se que a melhor compreensão do processo de dobramento de proteínas pode resultar em importantes avanços da medicina e o desenvolvimento de novos medicamentos. Porém, os métodos atuais que permitem determinar corretamente a estrutura tridimensional de uma proteína, como cristalografia por Raios-X e a Ressonância Magnética Nuclear, são muito custosos, possuindo um tempo de realização e confirmação bastante elevados. Por exemplo, o repositório de sequências de proteína UniProtKB/TrEMBL tem atualmente 18,5 milhões de registros (14 de dezembro de 2011), porém o *Protein Data Bank – PDB* tem apenas a estrutura

conformacional de apenas 78.628 proteínas (17 de janeiro de 2012).

A ciência da computação tem um papel importante no processo de predição de estrutura de proteínas (*Protein Structure Prediction – PSP*), propondo modelos para estudar este problema (LOPES, 2008). Para isto faz-se necessária a abstração das informações para o dobramento, através de um modelo factível do ponto de vista físico e químico, que não seja computacionalmente custoso.

Atualmente, a simulação de modelos computacionais que leve em conta todos os átomos de uma proteína é inviável, mesmo com os recursos computacionais mais poderosos (BENÍTEZ; LOPES, 2010). Consequentemente, vários modelos simplificados que abstraem a estrutura das proteínas têm sido propostos. Basicamente, existem dois tipos de representação de polipeptídeos, a analítica e a discreta. A representação analítica descreve todas as informações sobre os átomos que compõem as proteínas. Por outro lado, a representação discreta descreve uma proteína em um nível muito reduzido de detalhes.

Embora tais modelos discretos não sejam realistas, eles utilizam algumas propriedades bioquímicas dos aminoácidos, e sua simulação pode mostrar algumas características interessantes de proteínas reais. Estes também permitem uma exploração extensa do espaço de busca, gerando hipóteses que não podem ser obtidos por outras abordagens, mas que pode ser reproduzido experimentalmente ou através de simulações refinadas (DILL, 1999). Esta é uma motivação importante para o desenvolvimento de métodos computacionais para prever a estrutura de proteínas. O modelo computacional mais simples para o problema de PSP é conhecido como modelo Hidrofóbico-Polar (HP), tanto em duas (2D-HP) como em três (3D-HP) dimensões (DILL et al., 1995).

Apesar de simples, a abordagem computacional para procurar exhaustivamente uma solução para o PSP usando os modelos HP foi provado ser *NP*-completo (NGO; MARKS; KARPLUS, 1994). Portanto, este fato tem motivado o desenvolvimento e utilização de várias meta-heurísticas para lidar com o problema. Sendo assim, vários métodos computacionais já foram aplicados com o intuito de resolver o problema de predição de estruturas de proteínas, na busca de soluções ótimas ou sub-ótima.

2.3.1 O modelo AB-2D *off-lattice*

O modelo AB-2D *off-lattice* foi um dos primeiros modelos propostos para representação de estruturas de proteínas, apresentado por Stillinger e Head-Gordon (1995). Este modelo considera que as proteínas podem ser representadas em duas dimensões, sendo tratadas como

sequências formadas por duas espécies de monômeros: 'A' – aminoácidos hidrofóbicos – e 'B' – aminoácidos hidrofílicos. Apesar da simplicidade da representação da estrutura de uma proteína, este modelo é útil para verificar algumas das propriedades das proteínas reais.

Neste modelo, os monômeros possuem entre si uma distância de comprimento unitário, sendo que um monômero está conectado ao próximo em uma cadeia através de um laço que forma um ângulo com o laço anterior, constituindo uma estrutura chamada de *backbone*.

No modelo AB, uma proteína composta por n -monômeros, são necessários $n - 2$ ângulos para representar a estrutura. Estes ângulos (θ_i) são definidos no intervalo entre -180° e 180° . Ângulos iguais a zero ($\theta_i = 0$), representam que dois laços estão na mesma linha, ângulos menores que zero ($\theta_i < 0$) representam rotações no sentido anti-horário e ângulos maiores que zero ($\theta_i > 0$) representam rotações no sentido horário, conforme ilustrado na proteína hipotética da Figura 8, composta por sete aminoácidos.

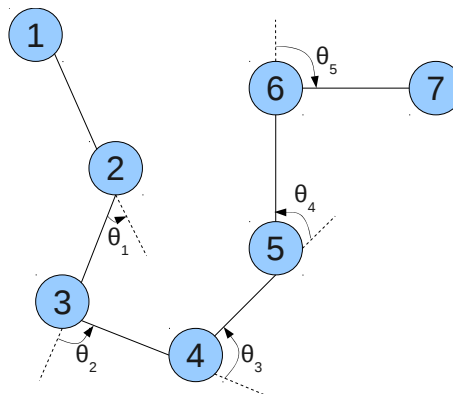


Figura 8 – Representação genérica a estrutura de uma proteína hipotética.
Fonte: Autoria própria.

Considerando dois monômeros genéricos i e j , a interação entre as espécies de monômeros (ξ_i e ξ_j) dá origem a diferentes valores de energia potencial (C). Valores positivos representam atração e negativos, repulsão: para ligações AA a energia é $+1$, os monômeros AA tendem a atrair fortemente um ao outro; ligações BB têm energia $+1/2$ e tendência de se atrair fracamente; e ligações AB ou BA têm energia $-1/2$ e têm a tendência de repulsão fraca. A energia de uma estrutura de proteínas com n monômeros é dada pela equação 4 (STILLINGER; HEAD-GORDON, 1995). Além disto, o modelo define valores de energia para os monômeros: 'A' possui energia 1 e 'B', energia -1 .

Esta equação postula dois tipos de energias potenciais intermoleculares através dos termos V_1 e V_2 . V_1 definido pela equação 5, depende apenas do ângulo entre monômeros para representar o potencial do *backbone*; V_2 , definido pela equação 6, representa a energia potencial presente nas interações não-conectadas e é conhecido como o potencial de Lennard-Jones.

A equação 7 representa a energia potencial decorrente da iteração entre os monômeros i e j , e d_{ij} é a distância entre estes monômeros na cadeia, tal que $i < j$.

$$\phi(\theta, \xi) = \sum_{i=1}^{n-2} V_1(\theta_i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^n V_2(d_{ij}, \xi_i, \xi_j) \quad (4)$$

$$V_1(\theta_i) = \frac{1}{4} \cdot (1 - \cos(\theta_i)) \quad (5)$$

$$V_2(d_{ij}, \xi_i, \xi_j) = 4 \cdot (d_{ij}^{-12} - C(\xi_i, \xi_j) \cdot d_{ij}^{-6}) \quad (6)$$

onde

$$C(\xi_i, \xi_j) = \frac{1}{8 \cdot (1 + \xi_i + \xi_j + 5 \cdot \xi_i \cdot \xi_j)} \quad (7)$$

Nos últimos anos, vários pesquisadores desenvolveram algoritmos para analisar este modelo (STILLINGER; HEAD-GORDON, 1995; HSU; MEHRA; GRASSBERGER, 2003; BACHMANN; ARKIN; JANKE, 2005; CHEN et al., 2011; KALEGARI, 2010). Hsu, Mehra e Grassberger (2003) testaram seu método em várias cadeias de Fibonacci com comprimento de até $N = 55$. Na Tabela 1 são apresentadas três destas sequências, as quais serão também utilizadas neste trabalho.

Tabela 1 – Instâncias do problema de dobramento de proteínas AB-2D

Aminoácidos	Sequência
13	ABBABBABABBAB
21	BABABBABABBABABBAB
34	ABBABBABABBABABBABABBABABBABABBAB

Fonte: (HSU; MEHRA; GRASSBERGER, 2003).

Estas sequências de Fibonacci são definidas recursivamente pela equação 8.

$$S_0 = A, S_1 = B, S_{i+1} = S_{i-1} * S_i \quad (8)$$

onde $*$ é o operador de concatenação e o comprimento das sequências é dado pelos números da sequência de Fibonacci⁶ (HSU; MEHRA; GRASSBERGER, 2003). As primeiras sequências são $S_2 = AB$, $S_3 = BAB$, $S_4 = ABBAB$, $S_5 = BABABBAB$, $S_6 = ABBABBABABBAB$. Observe, por exemplo, que a sequência S_4 é composta por S_2 concatenada com S_3 .

Diversos pesquisadores utilizaram estas sequências de aminoácidos. Destes, selecionamos dois em especial, com os resultados obtidos no trabalho inicial de Stillinger e Head-Gordon

⁶Sequência de Fibonacci é uma sucessão de números, onde os elementos são a soma dos seus dois antecessores e os dois primeiros números iguais a 1.

(1995) e as melhores soluções encontradas por Zhi-Peng, Wen-Qi e He (2007), utilizando o *Quasi-physical Algorithm*, os quais são apresentados na Tabela 2.

Tabela 2 – Alguns resultados encontrados na literatura para o problema de dobramento de proteínas AB-2D

Sequência	Stillinger	Zhi-Peng
13	-3,2245	-4,795
21	-5,2881	-12,326
34	-8,9749	-42,503

Fonte: Dados extraídos de Stillinger e Head-Gordon (1995) e Zhi-Peng, Wen-Qi e He (2007).

2.4 OTIMIZAÇÃO ESTRUTURAL DE TRELIÇAS

Segundo Bouzy e Abel (1995), treliças são uma forma de estrutura simples, mas amplamente utilizadas, que possuem um número limitado de graus de liberdade. Uma grande variedade de métodos tem sido desenvolvida ao longo dos anos para encontrar a configuração de tamanho padrão dos elementos que minimiza o peso de uma treliça com uma determinada geometria e topologia. Estes métodos têm como objetivo determinar os parâmetros de projeto, tais como as propriedades dos materiais, dimensionamento, forma e topologia, para alcançar um determinado objetivo.

Em engenharia estrutural e mecânica, um problema importante é determinar a resposta do sistema segundo os valores de entrada dados. No entanto, estes valores precisam satisfazer certos requisitos, como força e rigidez, para certificar-se se um sistema é seguro e útil.

Na otimização de treliças, três subproblemas diferentes podem estar envolvidos no projeto de otimização, cada subproblema com uma metodologia própria. A otimização dimensional, empregada neste trabalho, consiste no dimensionamento das seções transversais ótimas dos elementos da treliça ou determinação da geometria da estrutura. A otimização geométrica consiste na localização ótima dos nós da estrutura no espaço de coordenadas bidimensional ou tridimensional. Já a otimização topológica, define o número de nós e elementos da estrutura, definindo a conectividade entre eles.

Geralmente, o objetivo é minimizar o peso sujeito a restrições de deslocamento máximo e tensão admissível da estrutura, utilizando a área da seção transversal de cada elemento, bem como limites das próprias variáveis de projeto. O objetivo de minimizar o peso é conflitante com as restrições, pois se deseja reduzir o deslocamento e a tensão que uma treliça sofre, a área da seção transversal tem que ser aumentada, conseqüentemente, aumentando o peso da estrutura. Enquanto a tensão admissível e o peso costumam ter valores grandes, o deslocamento máximo permitido é, em geral, um valor pequeno.

A função objetivo do problema de otimização estrutural de treliças é dada pela Equação 9.

$$\min W = \sum_{i=1}^N \rho_i L_i A_i \quad (9)$$

onde A_i é a área da seção transversal de cada elemento i , L_i é o comprimento do elemento, ρ_i é a densidade do material e N é o número de elementos da treliça.

Além do peso, as restrições de deslocamento e tensão admissível são adicionadas ao valor da função objetivo como penalidade, afastando o resultado do ideal, proporcionalmente ao grau de violação das restrições.

A restrição de tensões normais é dada pela Equação 10.

$$\bar{\sigma}_i^c \leq \sigma_i \leq \bar{\sigma}_i^t \quad (10)$$

onde $\bar{\sigma}_i^c$ é a tensão admissível à compressão no elemento i , representado por valores negativos, $\bar{\sigma}_i^t$ é a tensão admissível à tração no elemento i , representado por valores positivos, σ_i é a tensão no elemento i para uma treliça analisada.

A restrição de tensões de flambagem é dada pela Equação 11.

$$\bar{\sigma}_i^b \leq \sigma_i \quad (11)$$

onde $\bar{\sigma}_i^b$ é a tensão admissível de flambagem de Euler no elemento i . O efeito de flambagem ocorre quando o elemento está sujeito à compressão, segundo a Equação 12.

$$\frac{\pi^2 \cdot E_i \cdot l_i}{A_i \cdot l_i^2} \leq \sigma_i \quad (12)$$

onde E_i é o módulo de elasticidade longitudinal do material do elemento i e l_i é o comprimento de flambagem dado por $l = \frac{L}{f}$, sendo f o fator de flambagem ($f = 1.0$ na maioria das barras de treliças) e I_i o menor momento de inércia da seção do elemento i para determinada treliça analisada.

A Equação 13 representa as restrições de deslocamento dos nós.

$$u_j^L \leq u_j \leq u_j^U \quad (13)$$

onde u_j^L é o limite inferior de deslocamento do nó j , u_j^U é o limite superior de desloca-

mento do nó j e u_j , o deslocamento do nó j para determinada treliça.

As restrições da área da seção transversal são dadas pela Equação 14.

$$A_i^L \leq A_i \leq A_i^U \quad (14)$$

onde A_i^L é o limite inferior da área da seção transversal do elemento i , A_i^U é o limite superior da área do elemento i e A_i é a área do elemento i para determinada treliça.

Na otimização deste problema através de meta-heurísticas as restrições, geralmente, são tratadas através de penalidades que modificam o valor da função objetivo.

Segundo Wang (2006), devido à simplicidade desta representação, grande sucesso foi alcançado na otimização do projeto de estruturas planas e espaciais de treliças, por vários trabalhos, como por exemplo Venkayya (1971), Haug e Arora (1979) e Kirsch (1993).

A seguir são apresentadas algumas instâncias de problemas de otimização estrutural de treliças, sendo elas, treliça plana de 10 barras e treliça plana de 200 barras.

2.4.1 Treliça Plana de 10 barras

A treliça de 10 barras é um problema de *benchmark* proposto por Berke e Khot (1974) e utilizado por diversos pesquisadores para testar os resultados obtidos por seus métodos de otimização, tais como Camp, Pezeshk e Cao (1998), Lee e Geem (2004). Neste problema existem 10 variáveis de projeto independentes, as quais são valores contínuos da seção transversal de cada elemento.

A geometria inicial da estrutura é mostrada na Figura 9. Esta estrutura é composta por 6 nós interconectados por 10 barras. Os nós 5 e 6 estão fixos, enquanto que os nós 1, 2, 3 e 4 estão livres, podendo sofrer ação das cargas P_1 e P_2 . Para efeito de comparação dos resultados obtidos nas referências serão mantidas as unidades em *kips* (unidade de força) e polegadas, não utilizando as unidades do Sistema Internacional (SI). A densidade do material é de $0,11b/pol^3$, o módulo de elasticidade é de $104ksi/pol^2$, e os membros são submetidos a limitações de tensão admissível de $\pm 25ksi$ (tração e compressão) e limitações de deslocamento de 2,0 polegadas que são impostas em cada nó em ambas as direções e área mínima de $0,1in^2$.

Existem dois modelos de carga para esse problema (BERKE; KHOT, 1974). No primeiro modelo é aplicada individualmente uma carga $P_1 = 100N$ para os nós 2 e 4. Já no segundo modelo os dois valores de carga são atribuídos, $P_1 = 100N$ sobre os nós 2 e 4 e $P_2 = 50N$ sobre os nós 1 e 3. Sendo o primeiro modelo de carga foi utilizado neste trabalho.

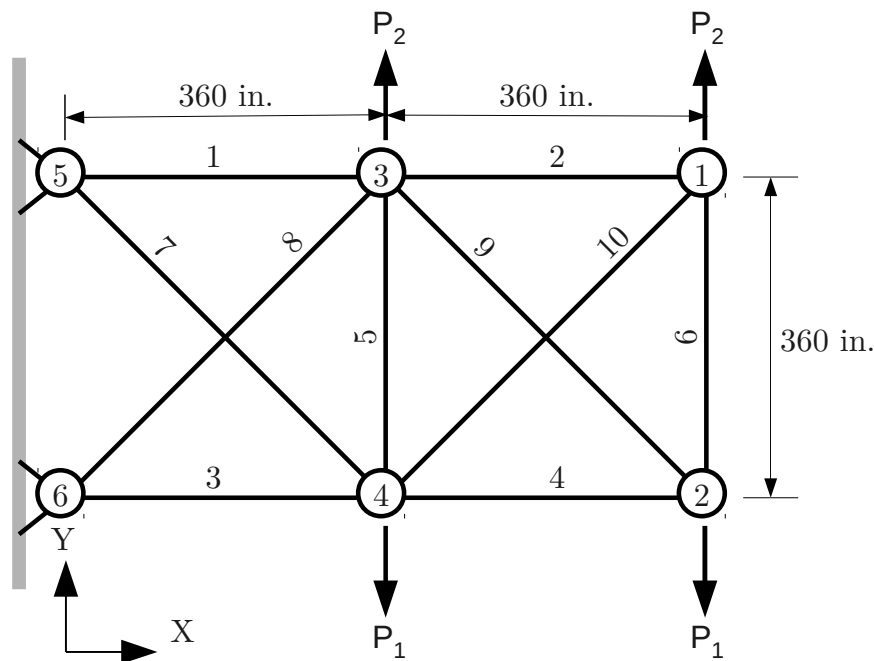


Figura 9 – Geometria inicial da treliça de 10 barras.
Fonte: Baseado em Camp, Pezeshk e Cao (1998), Lee e Geem (2004).

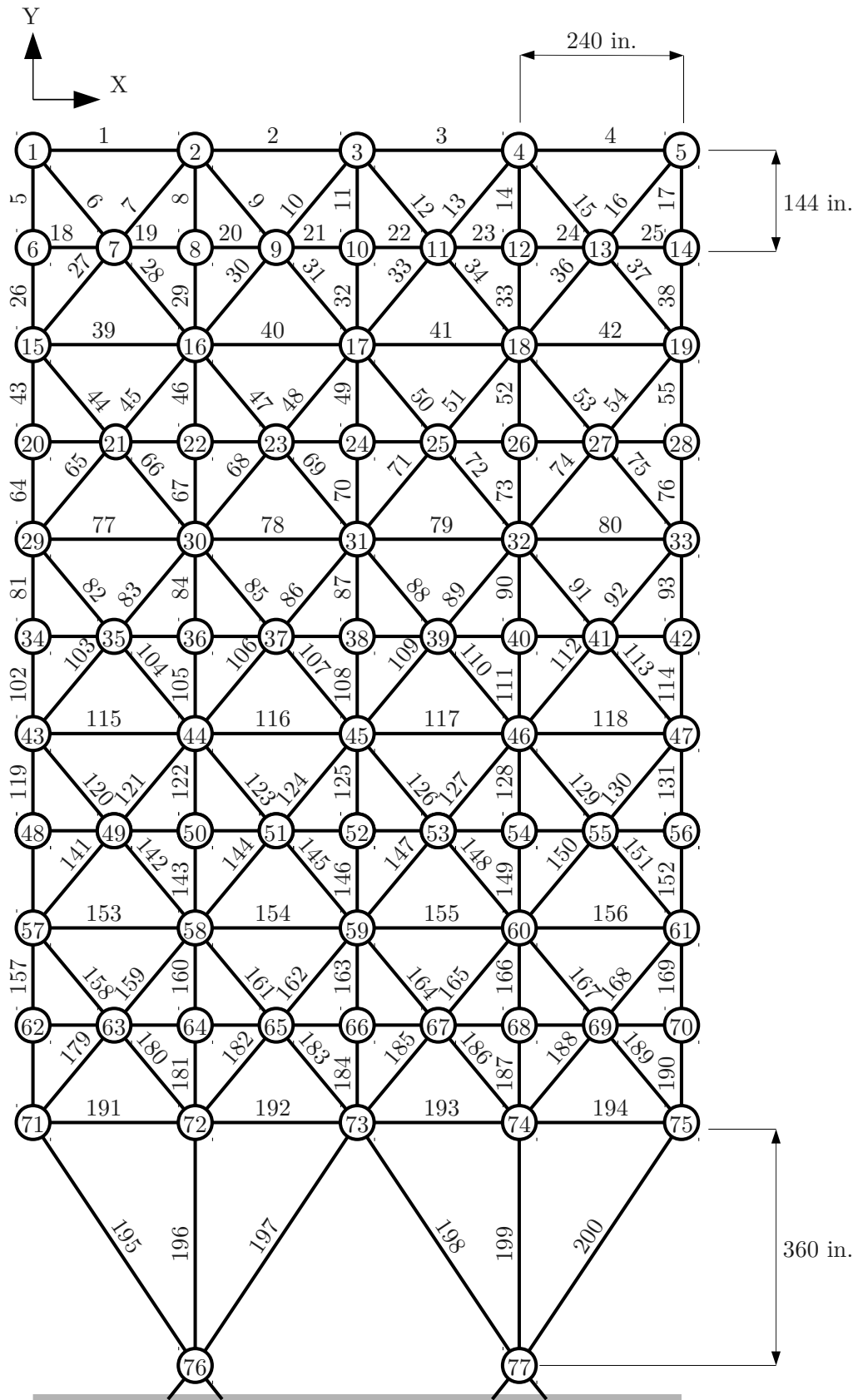
2.4.2 Treliça Plana de 200 barras

A treliça plana de 200 barras, mostrado na Figura 10, é um problema de otimização proposto por Venkayya, Khot e Reddy (1968) e utilizado como *benchmark* por diversos pesquisadores em métodos matemáticos e algoritmos evolucionários (COELLO; CHRISTIANSEN, 2000; LEE; GEEM, 2004; LAMBERTI, 2008).

Como no problema anterior, para efeito de comparação dos resultados obtidos nas referências as unidades são mantidas em kips e polegadas.

Para todos os casos são considerados tensão admissível para cada barra de 10ksi , limite de deslocamento de $\pm 0,5\text{polegadas}$ imposto a todos os nós livres, área mínima das barras de $0,01\text{pol}^2$, módulo de elasticidade do material 30.000ksi e densidade do material $0,283\text{lb/pol}^3$.

Esta estrutura de treliça é projetada usando diferentes tipos de restrições com números diferentes de variáveis de projeto. Existem três condições de carga para esta treliça (HAUG; ARORA, 1979), sendo que apenas a mais complexa é considerada neste trabalho. Consideramos, então, o caso de condição de carga no qual existe uma combinação de cargas no eixo X e Y . Carga de 1kip positiva atuando na direção X nos nós 1, 6, 15, 20, 29, 34, 43, 48, 57, 62 e 71; e carga de 10kips negativa, na direção Y nos nós 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20, 22, 24, 26, 28, 29, 30, 31, 32, 33, 34, 36, 38, 40, 42, 43, 44, 45, 46, 47, 48, 50, 52, 54, 56, 57, 58, 59, 60, 61, 62, 64, 66, 68, 70, 71, 72, 73, 74, e 75.



**Figura 10 – Geometria inicial da treliça de 200 barras.
Fonte: Baseado em Venkayya, Khot e Reddy (1968).**

Na implementação utilizada, os elementos da treliça são divididos em 29 grupos e cada grupo possui uma área de secção transversal que formam as 29 variáveis independentes de projeto dado por Coello e Christiansen (2000) e Lee e Geem (2004). A Tabela 3 apresenta a distribuição dos elementos nestes grupos.

Tabela 3 – Grupos de elementos para a treliça de 200 barras

Grupo	Elementos	Grupo	Elementos
1	1, 2, 3, 4	16	82, 83, 85, 86, 88, 89,91, 92, 103, 104, 106, 107,109, 110, 112, 113
2	5, 8, 11, 14, 17	17	115, 116, 117, 118
3	19, 20, 21, 22, 23, 24	18	119, 122, 125, 128, 131
4	18, 25, 56, 63, 94, 101, 132, 139, 170, 177	19	133, 134, 135, 136,137, 138
5	26,29,32,35,38	20	140, 143, 146, 149, 152
6	6, 7, 9, 10, 12, 13, 15, 16, 27, 28, 30, 31, 33, 34, 36, 37	21	120, 121, 123, 124, 126, 127,129, 130, 141, 142, 144, 145, 147, 148, 150, 151
7	39, 40, 41, 42	22	39, 40, 41, 42
8	43, 46, 49, 52, 55	23	43, 46, 49, 52, 55
9	57, 58, 59, 60, 61, 62	24	171, 172, 173, 174, 175, 176
10	64, 67, 70, 73, 76	25	178, 181, 184, 187, 190
11	44, 45, 47, 48, 50, 51, 53, 54, 65, 66, 68, 69, 71, 72, 74, 75	26	158, 159, 161, 162, 164, 165, 167, 168, 179, 180, 182, 183, 185, 186, 188, 189
12	77, 78, 79, 80	27	191, 192, 193, 194
13	81, 84, 87 90, 93	28	195, 197, 198, 200
14	95, 96, 97, 98, 99, 100	29	196, 199
15	102, 105, 108, 111, 114		

Fonte: Baseado em Coello e Christiansen (2000) e Lee e Geem (2004).

Alguns trabalhos como Lamberti e Pappalettere (2003) e Lamberti (2008) realizam experimentos sem o mapeamento de grupos, tratando cada um dos 200 elementos da treliça como uma variável independente de projeto.

2.5 PROBLEMAS DE OTIMIZAÇÃO DE FUNÇÃO

Os problemas de otimização de função são problemas puramente matemáticos utilizados para testar estratégias de otimização, nos quais procura-se encontrar valores extremos (ou maior ou menor) que a função pode assumir em um determinado intervalo.

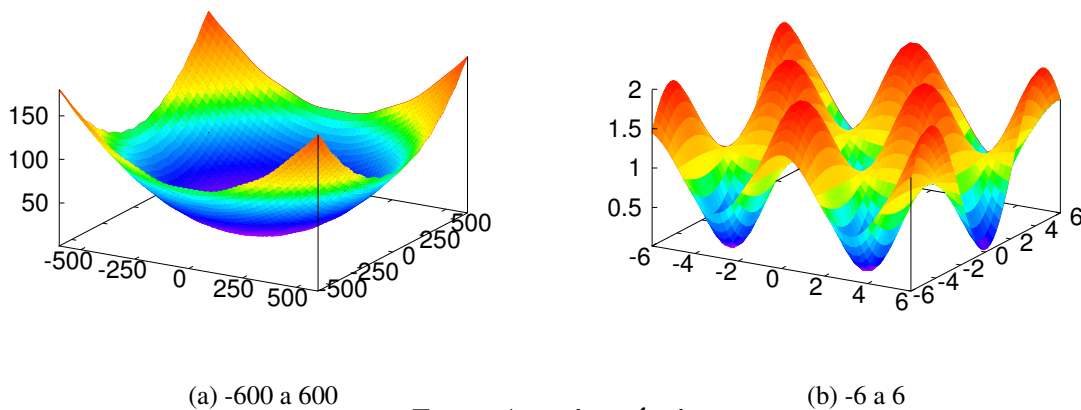
Segundo Cho, Olivera e Guikema (2008), a função de Griewank é uma função multimodal que tem sido amplamente utilizada para testar a convergência dos algoritmos de otimização, pois o número de mínimos cresce exponencialmente conforme aumenta o número de

dimensões. O intervalo de valores utilizados é de -600 a 600, sendo que o mínimo global está localizado em 0.

A função de Griewank é definida pela Equação 15. A Figura 11 ilustra esta função em duas dimensões, sendo que a primeira (11a) foi plotada no intervalo de -600 a 600, mostrando todo o espaço de busca e a segunda (11b) no intervalo de -6 a 6, mostrando em detalhes as oscilações de menor amplitude.

$$f(\vec{x}) = \sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos \frac{x_i}{\sqrt{i}} + 1 \quad (15)$$

Figura 11 – Função de Griewank.



Fonte: Autoria própria.

A função de Rosenbrock, ilustrado na Figura 12 em duas dimensões, é uma função caracterizada por um vale profundo que lembra uma parábola, levando ao mínimo global. O intervalo de busca para este problema é definido entre -5 e 5. Apesar de sua aparência simples, esta é uma função de difícil otimização, particularmente com muitas dimensões, pois no extenso platô é difícil obter uma indicação precisa da direção do ótimo global (DIGALAKIS; MARGARITIS, 2002). A função de Rosenbrock multidimensional é definida pela Equação 16.

$$f(\vec{x}) = \sum_{i=1}^{n-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2] \quad (16)$$

A função de Schaffer (Figura 13) é também uma função fortemente multimodal com muitos ótimos locais distribuídos e concêntricos ao ótimo global. Segundo Castro e Timmis (2002), o ótimo global é difícil de encontrar porque o valor dos ótimos locais diferem minimamente do ótimo global (em torno de 0,001). A função de Schaffer é definida pela Equação 17.

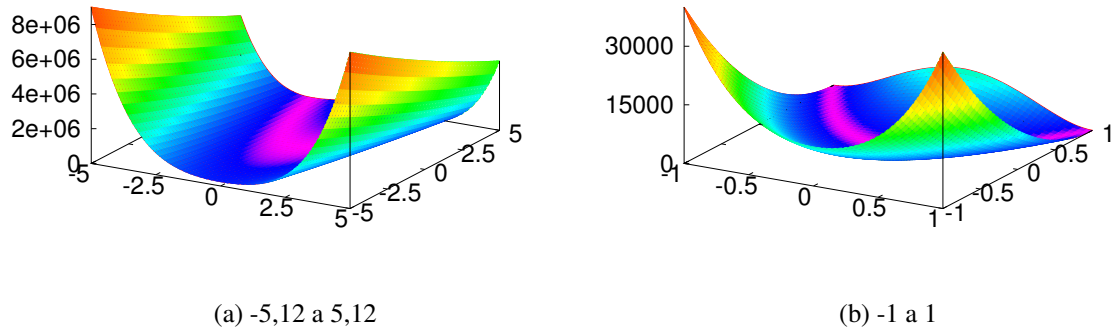


Figura 12 – Função de Rosenbrock.
Fonte: Autoria própria.

$$f(\vec{x}) = \sum_{i=1}^{n-1} \left[0,5 + \frac{\text{sen}(\sqrt{x_{i+1}^2 + x_i^2}) - 0,5}{(0,001 * (x_{i+1}^2 + x_i^2)^2 + 1)} \right] \quad (17)$$

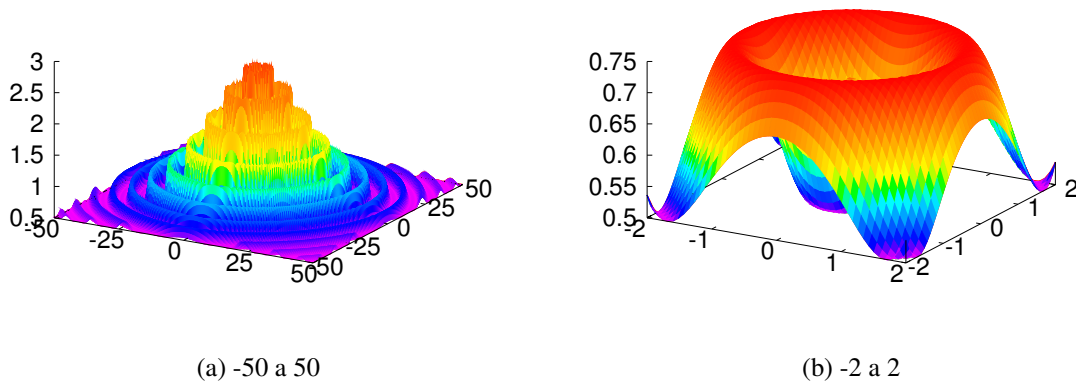


Figura 13 – Função de Schaffer.
Fonte: Autoria própria.

2.6 ANÁLISE ESTATÍSTICA

As informações contidas nesta seção são baseadas em LEVINE et al. (2008)

2.6.1 Análise de Variância

A análise de variância, ANOVA ou Teste F, é um método estatístico para se testar a igualdade de duas ou mais médias populacionais, de distribuição normal, através da análise

de variâncias amostrais. É aplicada quando é realizado um experimento que envolve amostras de mais de dois grupos (populações) e se necessita testar hipóteses. Considera-se a hipótese nula aquela que propõe que as médias estudadas são estatisticamente iguais contra a hipótese alternativa, propondo que pelo menos uma das médias é diferente das demais.

Esta análise é amplamente utilizada, pois pode ser aplicada a inúmeros casos de estudos estatísticos comparativos, ajudando a evitar armadilhas particulares, como rejeitar uma hipótese verdadeira, pelo uso de um teste que verifica a igualdade de várias médias.

Possui como suas variáveis: o fator da ANOVA, tomado como a característica do experimento, ou seja, a variável independente do estudo (x). Geralmente, este fator envolve diversos níveis, que são os tipos de tratamentos ou população que engloba cada fator; o nível de confiança (margem de erro do teste) deve ser previamente adotado, geralmente utilizado como 5%, o que influi nos resultados finais e conclusões, pois o teste de análise de variância utiliza como comparação a margem de erro adotada. Para realização do teste estatístico de análise de variância, as unidades experimentais devem ser homogêneas (com mesmas características e unidades de medida).

2.6.2 Teste de normalidade

A utilização da Análise de Variância deve ser feita para médias populacionais com distribuição normal. Para isto, se torna necessário o teste de normalidade das variáveis que determinam se um conjunto de dados aleatórios é modelado por uma distribuição normal.

A distribuição normal, também conhecida como distribuição Gaussiana, é uma das mais importantes distribuições da estatística. A Figura 14 apresenta os valores de probabilidade de ocorrência da distribuição normal de uma variável aleatória X , destacando o intervalo entre x_1 e x_2 , no qual encontram-se 68,2% das ocorrências.

É interessante salientar que em uma distribuição normal, o ponto mais alto da curva concentra a média, a mediana e a moda da distribuição, sendo a curva simétrica a este ponto. O desvio padrão determina a largura da curva, assim, valores maiores resultam em curvas mais largas e mais amplas, mostrando maior variabilidade dos dados.

Neste trabalho foi utilizado o método de Kolmogorov-Smirnov para determinar se os valores pertencem ou não a uma distribuição normal. O princípio básico do teste é verificar a hipótese (H_0) de que os dados seguem uma distribuição normal. Para isto, é medida a distância entre os valores aleatórios e uma distribuição normal correspondente.

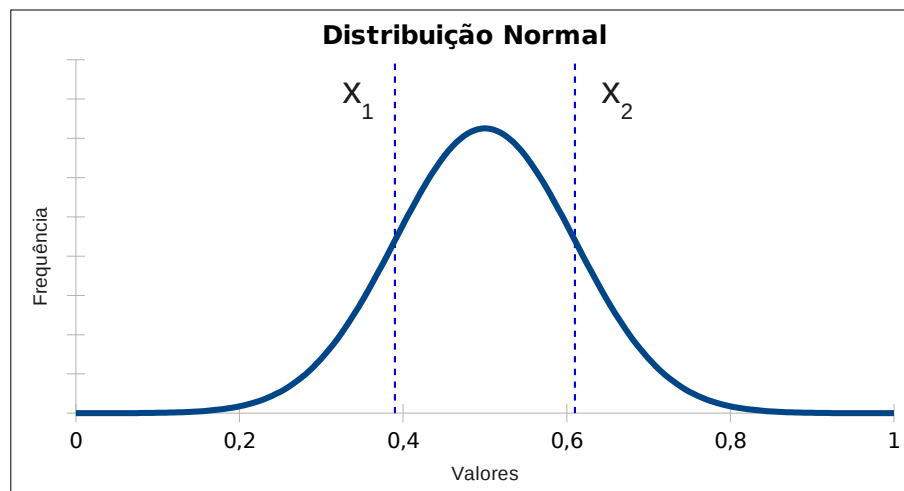


Figura 14 – Curva normal.
Fonte: Autoria própria.

2.6.3 Teste de Scott-Knott

O teste de Scott-Knott é um teste estatístico destinado a avaliação de agrupamentos. É tomado um determinado tratamento (variável independente) para ser analisado, separando experimentos influenciados por este tratamento em grupos que diferem estatisticamente entre si.

Segundo (SANTOS, 2000), apesar da existência de uma bibliografia bem formulada e vasta diversidade de programas computacionais para utilização de testes de comparações de médias, os mais comuns são o Teste de Tukey, Duncan, LSD, que apresentam resultados de difícil interpretação devido a sua grande ambiguidade. O teste Scott-Knott apresenta um método diferenciado de análise baseada em agrupamento univariada, eliminando o problema de ambiguidade pois os grupos são homogêneos e ocorre uma minimização da variação.

O Teste de Scott-Knott utiliza a verossimilhança para testar os tratamentos, sendo que estes são divididos em dois grupos que maximizem a soma de quadrados entre grupos, sendo que um número pequeno de tratamentos se torna mais fácil de obter os grupos, quando o número de tratamento é grande, o número de grupos cresce exponencialmente, o que dificulta a aplicação do teste, para minimizar este problema basta que as médias dos tratamentos sejam ordenadas (RAMALHO; FERREIRA; OLIVEIRA, 2005).

Scott-Knott (SCOTT; KNOTT, 1974), diz que uma vez que as médias estão ordenadas o procedimento ocorre de forma que:

(i) Determina-se a partição entre os dois grupos, visando a maximização da soma de quadrados entre os grupos.

(ii) Determinação do valor da estatística para uma máxima verossimilhança entre os tratamentos.

(iii) Feita a análise estatística, em que a máxima verossimilhança se já maior ou igual que a soma de quadrados entre grupos, rejeita-se a hipótese de que os dois grupos são idênticos e aceita-se a hipótese alternativa, em que os dois grupos são diferentes.

(iv) Caso a hipótese de rejeição desta hipótese, os dois grupos formados são independentemente submetidos aos passos (i) e (iii). Este processo é encerrado ao aceitar a hipótese nula no passo (iii) ou se cada grupo conter apenas uma média.

Podemos destacar que a principal vantagem do uso do teste de Scott-Knott é a ausência de ambiguidade nos procedimentos de comparação das médias, e esta funcionando muito bem para grande quantidade de tratamentos.

2.7 TRABALHOS RELACIONADOS

Com o surgimento de linguagens de programação destinadas ao desenvolvimento de aplicações para GPGPU, está ocorrendo um crescente interesse no uso de GPUs para processos de otimização. Devido à natureza implicitamente paralela, diversas meta-heurísticas, tais como Algoritmos Genéticos foram implementadas em arquiteturas paralelas. Um dos primeiros trabalhos aplicando GPU à otimização com o Algoritmos Genéticos foi proposto por Wong e colaboradores (WONG; WONG; FOK, 2005; WONG; WONG, 2006; FOK; WONG; WONG, 2007). Nos trabalhos de Wong apenas a avaliação da população foi executada na GPU. Como todo o processamento do AG permanecia em CPU, uma grande transferência de dados ocorria entre CPU e GPU. Outro trabalho que se destaca por seu pioneirismo na execução completa de uma meta-heurística em GPU foi proposto por Yu, Chen e Pan (2005). Como na época não havia uma plataforma de desenvolvimento para GPU, estes trabalhos foram desenvolvidos utilizando bibliotecas como Direct3D e OpenGL.

Com o surgimento da plataforma CUDA, a programação em GPU se tornou mais acessível, possuindo uma abordagem mais simples baseada em *threads*. Fazendo o uso destes recursos, Zhu (2009) apresentou uma implementação de um algoritmo de estratégia de evolução para resolver diversos problema contínuos. Em sua implementação são projetados vários *kernels* para a execução dos operadores de seleção, *crossover* e mutação, além de executar a avaliação paralela dos indivíduos da população, permanecendo o controle de fluxo em CPU. Arora, Tulshyan e Deb (2010) e Yoshimi et al. (2010) apresentaram implementações similares investigando a influência do número de *threads* e do tamanho da população sobre o

desempenho da GPU.

Zhu, Xiao e Gu (2010) apresentaram uma paralelização do algoritmo de seleção clonal para solucionar o problema de dobramento de proteínas AB-2D *off-lattice* utilizando a plataforma CUDA. É apresentada uma nova visão na pesquisa de predição de estrutura de proteínas, através da redução do tempo de processamento, mantendo uma precisão aceitável.

3 METODOLOGIA

3.1 DESCRIÇÃO DO TRABALHO

Neste capítulo é apresentada a proposta de modificação da meta-heurística Busca Harmônica. Primeiramente, propõe-se a inclusão de população no algoritmo, além da aplicação de ideias baseadas em outras meta-heurísticas. Em seguida, é realizada a paralelização do modelo proposto utilizando uma placa de processamento gráfico com tecnologia CUDA.

3.2 IMPLEMENTAÇÃO DA BUSCA HARMÔNICA

A implementação do algoritmo Busca Harmônica foi realizada em forma de *framework* utilizando linguagem C. Uma estrutura simples foi definida para desenvolvimento na qual o usuário necessita implementar apenas sua função objetivo e configurar um arquivo contendo os parâmetros do algoritmo.

Os parâmetros de configuração do algoritmo são informados no arquivo "input.file". Neste são definidos os parâmetros: *musicians* que representa o número de variáveis do problema, equivalente ao parâmetro N da HS; *plays* que representa o número máximo de iterações do algoritmo (MI); *harmony_memory_size* que representa o tamanho da memória harmônica (HMS); *harmony_memory_consideration* que representa a taxa de consideração da memória harmônica ($HMCR$); *pitch_adjusting* que representa a taxa de ajuste fino (PAR); *pitch_value* que representa o tamanho máximo do ajuste fino (FW);

A implementação da função objetivo é realizada a partir do protótipo de um método definido por `objective_function(harmony h)` onde *harmony* é uma estrutura composta pelo vetor *musicians*, representando as variáveis de entrada, *neval* que armazena o número da avaliação e aptidão (*fitness*) representando o valor calculado da função objetivo, sendo atribuído ao término da execução do método.

3.3 BUSCA HARMÔNICA BASEADA EM GERAÇÕES

Buscando-se melhorar o desempenho do algoritmo Busca Harmônica com o uso de arquiteturas paralelas, foi desenvolvido a Busca Harmônica baseada em população (*Population-Based Harmony Search* – PBHS). Na PBHS é proposto o uso de uma população de harmonias temporárias, denominadas Arranjos Musicais (AM). Neste caso, os Arranjos Musicais são uma extensão da Memória Harmônica, sendo que eles são incluídos no final desta, formando um único conjunto de tamanho $HMS + PS$.

O conjunto composto pela união da Memória Harmônica com os Arranjos Musicais é considerado por completo no processo de improviso para gerar novas harmonias. A cada iteração, várias harmonias novas são improvisadas e incluídas nos Arranjos Musicais. No final de cada ciclo, após a avaliação das novas harmonias geradas, apenas a melhor dentre os Arranjos Musicais participará efetivamente do processo de atualização.

O Algoritmo 3 mostra a PBHS, destacando (nas linhas sublinhadas) as diferenças com a HS. Como na Busca Harmônica original, o processo inicia-se com o carregamento dos parâmetros e a definição da função objetivo. Na primeira linha do Algoritmo 3, um novo parâmetro ganha destaque, *PS* (Tamanho da População, do inglês *Population Size*), que define o número de harmonias dentre os Arranjos Musicais.

Na inicialização da memória harmônica (linha 4), as posições de memória referentes aos Arranjos Musicais passam a ser consideradas como uma posição da Memória Harmônica. Sendo assim, todas estas posições recebem uma harmonia durante a inicialização.

Na linha 8, o processo de improviso é repetido *PS* vezes, gerando uma nova harmonia para cada posição no espaço destinado aos Arranjos Musicais. Assim, uma população de harmonias temporárias é improvisada a cada iteração.

Durante o improviso, na linha 11, o tamanho da população é somado ao tamanho da Memória Harmônica. Assim, as posições da população são incluídas para a composição das novas harmonias. Conseqüentemente, indivíduos que não fazem parte da elite presente na Memória Harmônica podem ser usados para gerar novas harmonias válidas.

O uso da população de Arranjos Musicais tem como objetivo aumentar a característica exploratória do algoritmo, que, em algumas situações, pode necessitar ser compensada para que ocorra uma convergência em tempo hábil. Para intensificar a busca local, a seleção dos indivíduos deixa de ser aleatória e passa a ser realizada por um torneio estocástico (linha 11), o qual é explicado na seção 3.5.1, reforçando a característica elitista do modelo proposto.

Algoritmo 3 Pseudo-código do algoritmo Busca Harmônica baseada em Geração.

```

1: Parâmetros: HMS, HMCR, PAR, MI, FW, PS {Tamanho da População}
2: Início
3: Função Objetivo  $f(\vec{x})$ ,  $\vec{x} = [x_1, x_2, \dots, x_N]$ 
4: Inicializa a Memória Harmônica e os Arranjos Musicais  $x^i$ ,  $i = 1, 2, \dots, (HMS+PS)$ 
5: Avalia cada Harmonia na HM:  $f(x^i)$ 
6: ciclo  $\leftarrow 1$ 
7: Enquanto ciclo < MI Faça
8:   Para new  $\leftarrow 1$  até PS Faça
9:     Para j  $\leftarrow 1$  até N Faça
10:      Se aleatório  $\leq$  HMCR Então {Taxa de Consideração da Memória}
11:         $x_j^{(HMS+new)} \leftarrow x_j^i$ , com  $i \in [1, (HMS + PS)]$ 
12:        { $i$  selecionado por torneio estocástico}
13:      Se aleatório  $\leq$  PAR Então {Taxa de Ajuste Fino}
14:         $x_j^{(HMS+new)} \leftarrow x_j^{(HMS+new)} \pm r \times FW$  {com  $r$  aleatório}
15:      Fim Se
16:      Senão {Seleção Aleatória}
17:        Gera  $x_j^{(HMS+new)}$  aleatoriamente
18:      Fim Se
19:    Fim Para
20:  Fim Para
21:  Avalia todas as novas harmonias geradas:  $f(x^i)$ ,  $i = HMS + (1, 2, \dots, PS)$ 
22:  Seleciona a melhor harmonia  $\vec{k}$  gerada dentre os Arranjos Musicais
23:  Se  $f(k)$  é melhor que a pior harmonia da HM Então
24:    Atualiza a Memória Harmônica
25:  Fim Se
26:  ciclo  $\leftarrow$  ciclo + 1
27: Fim Enquanto
28: Exibição dos Resultados
29: Fim

```

As novas harmonias improvisadas são avaliadas e incluídas no espaço destinado aos Arranjos Musicais (linha 20). Finalmente, a melhor harmonia dentre os Arranjos Musicais é selecionada para ser utilizada no processo de atualização (linhas 21 a 24).

3.4 MEGA BUSCA HARMÔNICA

O Mega Busca Harmônica (*Mega Harmony Search* – MHS) é a implementação da Busca Harmônica baseada em gerações executada em GPU e acrescida de algumas estratégias inspiradas em outras meta-heurísticas. Nesta implementação, todos os passos do algoritmo são executados em GPU, possibilitando a execução paralela de grande parte do algoritmo. Estando todos os passos em GPU, é possível um acesso mais rápido aos dados em memória, evitando transferências de dados desnecessárias entre a memória da CPU e a memória do dispositivo GPU.

A CPU fica encarregada apenas da inicialização dos parâmetros, alocação de posições da memória global da GPU e controle das iterações, além de copiar os valores iniciais necessários para a execução da MHS. O Algoritmo 9, apresentado no Apêndice B, mostra o fluxo principal de controle do algoritmo MHS implementado em linguagem C. Este fluxo é semelhante aos passos do HS apresentado anteriormente na Figura 5 na Seção 2.2.1.

A seguir são apresentados os passos da MHS executados em GPU e respectivas descrições de seus funcionamentos.

A inicialização da Memória Harmônica foi implementada com dois *kernels* diferentes, os quais são apresentados nos Algoritmos 10 e 11 do Apêndice B. O primeiro tem dois níveis de paralelismo, distribuindo cada posição da Memória Harmônica e dos Arranjos Musicais para uma *thread*, conforme ilustra a Figura 15. Cada índice de *thread* (T_i) aponta para valores de uma única harmonia, sendo cada bloco (B_i) responsável por manipular a mesma posição do vetor solução em todas as harmonias da Memória Harmônica. Neste *kernel* cada *thread* atribui um valor aleatório dentro dos limites permitidos (l_{min} e l_{max} , que são os limites inferior e superior, respectivamente) à sua variável e as *threads* do bloco de índice zero iniciam o vetor que armazena os valores da função objetivo com um valor grande (10^{20}) para todas as suas posições. Uma variável `device_random` armazena uma lista de valores aleatórios gerados através do método *Mersenne Twister*¹. Esta variável é utilizada por todos os *kernels* que necessitam de valores aleatórios é acompanhada de um índice único referenciado por `random_index`. Ao término da execução do *kernel* que faz uso dos valores aleatórios, o índice é atualizado. As-

¹*Mersenne Twister* é um gerador de números pseudo-aleatórios de alta qualidade, desenvolvido em 1997 por Makoto Matsumoto e Nishimura Takuji.

sim, os valores aleatórios são recuperados na mesma sequência em que foram gerados. Quando se esgotam os valores, um novo conjunto de valores é gerado.

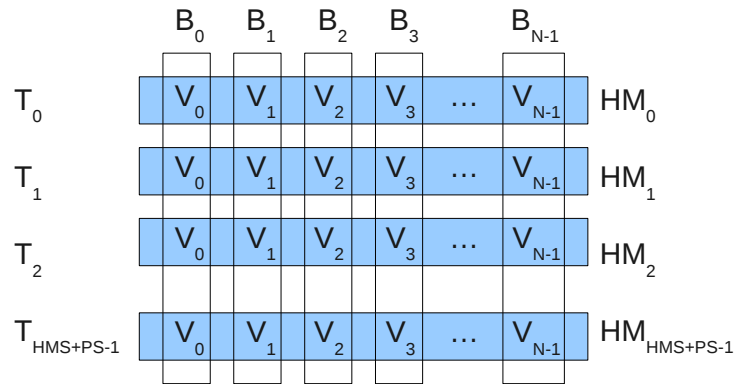


Figura 15 – Distribuição de threads (T) e blocos (B) no primeiro kernel do processo de inicialização.

Fonte: Autoria própria.

O segundo *kernel* é executado para cada posição da Memória Harmônica, produzindo uma nova harmonia para cada posição, como é ilustrado na Figura 16. Cada variável da nova harmonia é independente das demais. Desta forma, a obtenção do seu valor é realizada por N blocos (B_j) diferentes com um único índice de threads (T_{HMS}), sendo N o número de variáveis da harmonia. Após a geração de cada harmonia, a mesma é avaliada e inserida na Memória Harmônica através do processo de atualização.

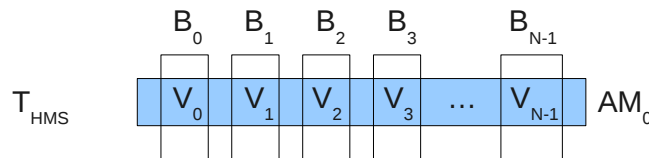


Figura 16 – Distribuição de threads (T) e blocos (B) no segundo kernel do processo de inicialização.

Fonte: Autoria própria.

Na GPU, a Memória Harmônica e os Arranjos Musicais são alocados na memória global e são representados linearmente, de forma que a cada N posições encontra-se uma harmonia diferente.

Depois que a Memória Harmônica for carregada com suas harmonias iniciais, começa o processo iterativo de otimização. A cada ciclo da iteração, três chamadas de *kernel* são realizadas. A primeira delas realiza o improviso das novas harmonias, a segunda, a seleção da melhor solução gerada e a terceira chamada, a atualização da memória harmônica.

No improviso, o processo de seleção de cada variável da nova harmonia é realizado independentemente, como ilustrado na Figura 17. Para realizar o improviso de uma nova har-

monia para cada posição dos Arranjos Musicais, cada índice de *thread* (T_i) fica responsável por uma harmonia diferente e cada bloco (B_i) responsável por uma variável de todas as novas harmonias. Logo, o número de blocos é igual ao número de variáveis do problema, e o número de *threads* é igual ao tamanho da população (PS). Criadas as novas harmonias, a avaliação da função objetivo das mesmas é realizada simultaneamente em paralelo. O Algoritmo 12 do Apêndice B apresenta o código deste processo.

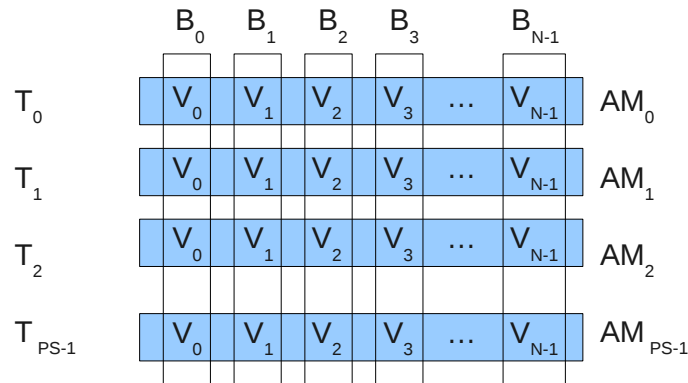


Figura 17 – Distribuição de *threads* (T) e blocos (B) no processo de improviso.
Fonte: Autoria própria.

Após a avaliação das harmonias geradas, um *kernel* com uma única *thread* realiza uma verificação sequencial, comparando as novas harmonias entre si selecionando a melhor. Caso a harmonia selecionada seja melhor do que a pior solução encontrada na Memória Harmônica, acontece a substituição desta.

A atualização da Memória Harmônica foi implementada em duas etapas:

1. A primeira etapa localiza a posição de inserção da nova harmonia, realizando uma busca sequencial na Memória Harmônica, procurando a primeira posição em que o valor da função objetivo da harmonia é pior que o da nova harmonia.
2. A segunda etapa, apresentada no Algoritmo 13 do Apêndice B, utiliza-se de dois níveis de paralelismo, de forma que cada bloco fique responsável por uma variável da nova harmonia e as *threads* fiquem responsáveis pelo processo de inserção e deslocamento na memória. Como ilustra a Figura 18. Sendo assim, o número de blocos é igual ao número de variáveis do problema (N) e o número de *threads* é igual ao número de harmonias na Memória Harmônica menos um ($HMS - 1$).

A atualização da Memória Harmônica inicia com o armazenamento das informações da Memória Harmônica para cada *thread*. As *threads* com índice menor do que a posição de inserção da nova harmonia não realizam processamento significativo. A *thread* com índice

igual à posição de inserção atribui a nova harmonia à sua posição de destino. As *threads* com índice maior que a posição realizam os deslocamentos das harmonias para as posições seguintes, mantendo a memória ordenada. Este processo de deslocamento e inserção também pode ser observado na Figura 18.

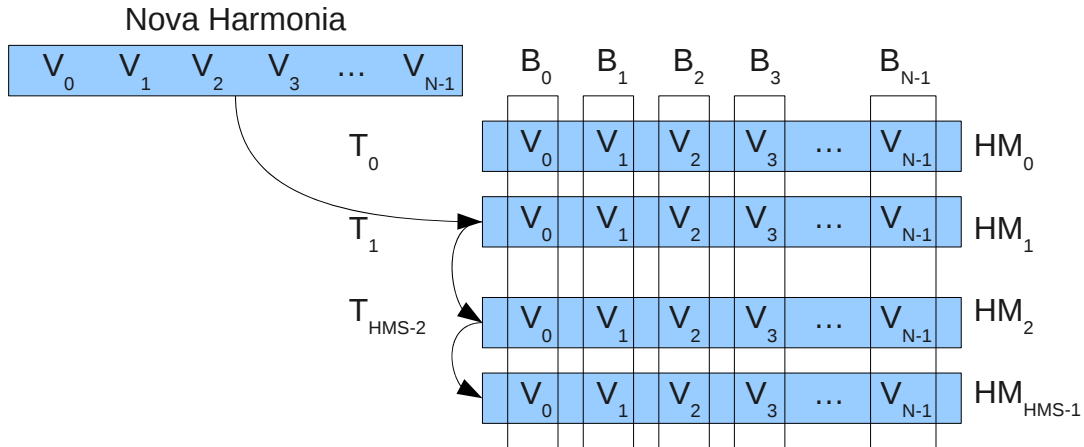


Figura 18 – Distribuição de *threads* (T) e blocos (B) no processo de atualização.
Fonte: Autoria própria.

Com a Memória Harmônica ordenada tem-se na primeira posição a melhor harmonia e, na última, a pior. Sendo assim, a atualização da Memória Harmônica é realizada inserindo a nova harmonia na sua devida posição, deslocando-se as posições subseqüentes e descartando-se a pior solução.

Concluída a execução do algoritmo, os dados referentes à melhor harmonia encontrada são transferidos do dispositivo GPU para a CPU para que possam ser utilizados e visualizados.

3.5 ESTRATÉGIAS PROPOSTAS

Algumas estratégias foram propostas com o intuito de melhorar a qualidade das soluções encontradas através do PBHS. Estas modificações são inspiradas em características de outras meta-heurísticas, tais como Algoritmos Genéticos e Otimização por Enxame de Partículas, além de novas ideias para a obtenção de melhor desempenho.

A combinação destas características pode apresentar melhorias significativas na qualidade de solução e até mesmo melhorar a característica de convergência, obtendo-se soluções de melhor qualidade com um número menor de avaliações de função, ou, também, evitando convergência prematura.

As melhorias propostas são apresentadas a seguir, sendo elas: a utilização de um método de seleção, blocos construtivos maiores, a utilização de uma harmonia base, a explosão

ou dizimação e a autoadaptação de parâmetros. O Algoritmo 4 apresenta como estão dispostas estas melhorias propostas durante o processo de otimização.

3.5.1 Método de seleção

A Busca Harmônica original não apresenta nenhum método de seleção para a escolha dos indivíduos que farão parte da nova harmonia. Logo, a utilização de um método de seleção pode dar um direcionamento melhor para o processo de evolução para a obtenção de melhores soluções, através de um processo de intensificação (linhas 10 e 18 no Algoritmo 4). Como a Busca Harmônica é um processo de otimização estocástica, um método de seleção que se mostra interessante para ser utilizado com a PBHS é o Torneio Estocástico, por ser um método de seleção pouco agressivo, permitindo a seleção tanto de indivíduos bons como potencialmente bons.

O Torneio Estocástico é um método de seleção no qual alguns indivíduos competem entre si, sendo que o indivíduo vitorioso é mantido para a próxima competição, o último indivíduo vencedor é o selecionado para participar do processo de geração (GOLDBERG, 1989). Testes preliminares mostraram que o uso de um método de seleção combinado com a PBHS pode levar a soluções de melhor qualidade. Este processo é ilustrado na Figura 19, onde é apresentado um torneio estocástico de tamanho 5, onde acontece a competição entre 5 indivíduos.

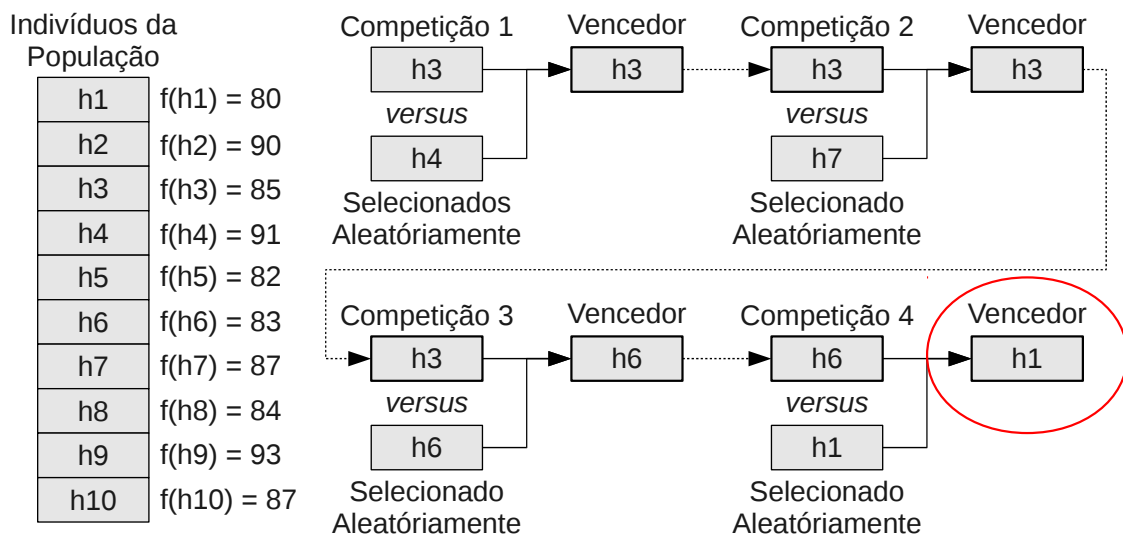


Figura 19 – Torneio Estocástico de tamanho 5.
Fonte: Autoria própria.

O número de indivíduos que participarão do torneio estocástico é definido pelo parâmetro TS (*Tourney Size*).

Algoritmo 4 Pseudo-código do algoritmo da MHS, incluindo as melhorias propostas.

```

1: Parâmetros: HMS, HMCR, PAR, MI, FW, PS {Tamanho da População}
2: Start
3: Função Objetivo  $f(\vec{x})$ ,  $\vec{x} = [x_1, x_2, \dots, x_N]$ 
4: Inicializa a Memória Harmônica e os Arranjos Musicais  $x^i$ ,  $i = 1, 2, \dots, (HMS+PS)$ 
5: Avalia cada Harmonia na HM:  $f(x^i)$ 
6: ciclo  $\leftarrow$  1; avaliacoes_sem_melhoria  $\leftarrow$  0
7: Enquanto ciclo < MI Faça
8:   Para new  $\leftarrow$  1 até PS Faça
9:     Carrega os parâmetros da autoadaptação
10:    HB = torneio_estocástico(TS) {seleciona o harmonia base}
11:    Para j  $\leftarrow$  1 até N Faça
12:      Se  $u(0, 1) \leq HMCR$  Então {Taxa de Consideração da Memória}
13:        BL = MBL * u(0, 1) {tamanho do bloco construtivo}
14:        Para b  $\leftarrow$  1 até BL Faça
15:          Se  $u(0,1) < UBHR$  Então {Verifica vai usar a harmonia base}
16:            i  $\leftarrow$  HB
17:            Senão
18:              i  $\leftarrow$  torneio_estocástico(TS)
19:            Fim Se
20:             $x'_j \leftarrow x^i$ 
21:            Se  $u(0, 1) \leq PAR$  Então {Taxa de Ajuste Fino}
22:               $x'_j \leftarrow x_j^{(HMS+new)} \pm u(-1, 1) \times FW$ 
23:            Fim Se
24:            j++
25:          Fim Para
26:          Senão {Seleção Aleatória}
27:            Gera  $x'_j$  aleatoriamente
28:          Fim Se
29:        Fim Para
30:      Fim Para
31:      Avalia todas as novas harmonias geradas:  $f(x^i)$ ,  $i = HMS + (1, 2, \dots, PS)$ 
32:      Seleciona a melhor harmonia  $\vec{k}$  gerada dentre os Arranjos Musicais
33:      Se  $f(k)$  é melhor que a pior harmonia da HM Então
34:        Atualiza a Memória Harmônica
35:        avaliacoes_sem_melhoria++
36:        Se  $\vec{k}$  substituiu a melhor harmonia da memória harmônica Então
37:          avaliacoes_sem_melhoria  $\leftarrow$  0
38:        Fim Se
39:      Fim Se
40:      ciclo  $\leftarrow$  ciclo + 1
41:      Se avaliacoes_sem_melhoria > EXP Então
42:        explosão(); avaliacoes_sem_melhoria  $\leftarrow$  0
43:      Fim Se
44:    Fim Enquanto
45:    Exibição dos Resultados
46:  End

```

3.5.2 Blocos construtivos maiores

Blocos construtivos são conjuntos de variáveis subsequentes que têm origem de um mesmo vetor solução. Originalmente, no algoritmo Busca Harmônica, as variáveis subsequentes de um novo indivíduo pertencem a indivíduos selecionados aleatoriamente. Sendo assim, há uma grande possibilidade de que estas pertençam a indivíduos diferentes.

O uso de blocos construtivos maiores intensifica o processo de busca local, tornando o processo de improviso da HS semelhante ao processo de *crossover* dos Algoritmos Genéticos. Isto reduz o número de indivíduos que participam do processo de improviso das novas harmonias, até casos em que apenas dois indivíduos participam do processo, tornando-o idêntico ao *crossover* de um ponto.

O parâmetro MBL (*Maximum Block Length*) define o comprimento máximo de cada bloco construtivo, sendo que a cada indivíduo selecionado é obtido aleatoriamente o tamanho do bloco construtivo que será copiado para a nova harmonia. O tamanho tem um valor aleatório entre 1 e MBL (linhas 13 e 14 no Algoritmo 4). A Figura 20 mostra um exemplo de blocos construtivos maiores com MBL = 3, onde até 3 variáveis subsequentes do mesmo indivíduo são utilizados para compôr um novo indivíduo (harmonia).

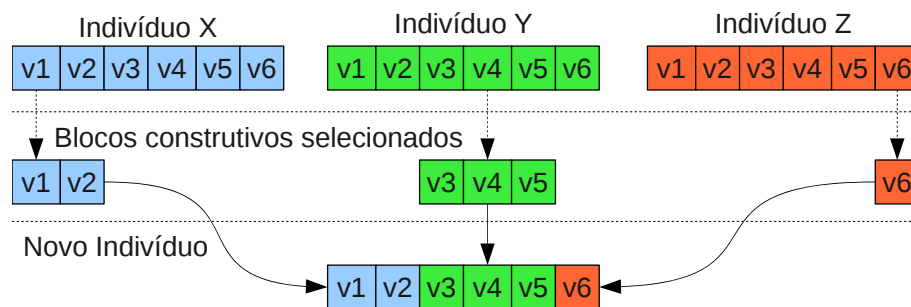


Figura 20 – Exemplo do uso de blocos construtivos maiores na HS.
Fonte: Autoria própria.

3.5.3 Harmonia base

Harmonia base é uma harmonia que é selecionada antes de iniciar o processo de improvviso. Um parâmetro define a taxa do uso da harmonia base (*Usage Base Harmony Rate – UBHR*). Um valor aleatório é comparado a UBHR e, caso seja menor, no momento do improvviso em que seria selecionada uma harmonia, o valor da harmonia base é utilizada (linhas 10, 15 e 16 no Algoritmo 4). Desta forma, assim como no caso dos blocos construtivos maiores, o uso de uma harmonia base reduz o número de harmonias na composição de novas harmonias. A Figura 21 apresenta um exemplo da utilização da harmonia base.

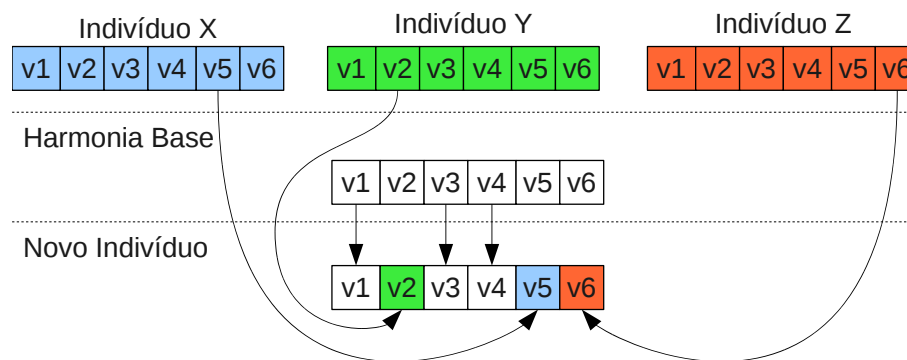


Figura 21 – Exemplo do uso harmonia base na HS.
Fonte: Autoria própria.

As novas harmonias geradas ficam mais próximas de uma determinada região, intensificando o processo de busca em uma determinada região do espaço de busca.

3.5.4 Explosão ou Dizimação

Dependendo dos parâmetros selecionados, ao alcançar um máximo local, o algoritmo pode ficar estagnado, perdendo rapidamente a diversidade dos indivíduos armazenados. Este fenômeno é conhecido como convergência prematura, impedindo que ocorram possíveis melhorias. Para solucionar este problema foi implementada uma estratégia conhecida como explosão ou dizimação.

Explosão ou dizimação é uma estratégia determinística que consiste na eliminação de um número fixo de indivíduos que possuem baixa aptidão (MOGNON, 2004). A posição ocupada pelos indivíduos eliminados é ocupada por novos indivíduos gerados aleatoriamente. Como a memória harmônica tende a apresentar uma diversidade muito baixa, com indivíduos muito semelhantes quando a estratégia é ativada, optou-se por manter apenas o melhor indivíduo, eliminando todos os demais. Nesta situação, sem a explosão, apenas um evento do

acaso exploraria outra região do espaço de busca, através de uma variação significativa no indivíduo.

Esta estratégia é de simples implementação. Porém, possui como desvantagem potencial eventuais remoções de indivíduos com características únicas, que poderiam levar o processo de otimização para regiões de ótimo ou sub-ótimo.

Como a HS é uma meta-heurística evolucionária baseada em um princípio elitista, o melhor indivíduo é mantido sempre na memória harmônica. Se após várias iterações o melhor indivíduo permanecer o mesmo, esta é uma grande evidência de que o algoritmo tenha convergido prematuramente. Esta estratégia verifica o melhor indivíduo a cada iteração. Caso ele não tenha alterado, um contador é incrementado de 1. Caso contrário, o mesmo contador é zerado. Quando o contador atingir um valor máximo, a estratégia de explosão é ativada (linhas 6, 35 a 38 e 41 a 43 do Algoritmo 4).

O uso de um processo de explosão combinado com a MHS permite renovar o processo de busca explorando novas regiões no espaço de busca, quando a evolução estiver estagnada por um certo número de avaliações.

Dois parâmetros configuram este procedimento, o número de sobreviventes (GAP) e o número de avaliação sem substituição do melhor indivíduo (EXP).

3.5.5 Autoadaptação de Parâmetros

Em toda meta-heurística, a escolha adequada de parâmetros é uma tarefa difícil. Segundo (KITA, 2001), a adaptação automática de parâmetros simplifica a configuração inicial de meta-heurísticas evolucionárias, não havendo a necessidade da configuração de alguns parâmetros de controle, os quais são adaptados pelo próprio método durante o processo de evolução, simplificando o uso da Busca Harmônica, podendo obter soluções semelhantes às obtidas com o uso de parâmetros de controle pré-definidos.

Os parâmetros que mais influenciam o processo de otimização da Busca Harmônica são *HMCR* e *PAR*, que equilibram a busca local e global. Sendo assim, estes foram selecionados para o processo de autoadaptação. A autoadaptação na Busca Harmônica foi implementada incluindo-se duas novas variáveis às harmonias, contendo o par de valores de *HMCR* e *PAR*.

Optou-se por realizar a atualização dos parâmetros a cada 10 iterações do algoritmo (linha 9 no Algoritmo 4), antes do improvisto das novas harmonias. Duas diferentes estratégias de autoadaptação foram implementadas. Na primeira, identificada por A0, os valores de *HMCR* e *PAR* são calculados pela média dos valores armazenados na Memória Harmônica. Na segunda,

identificada por A1, cada posição dos Arranjos Musicais possui um *HMCR* e *PAR* únicos que são utilizados no processo de improviso para gerar o novo indivíduo que ocupará sua respectiva posição.

4 RESULTADOS EXPERIMENTAIS

4.1 ORGANIZAÇÃO DOS EXPERIMENTOS

A plataforma experimental utilizada neste trabalho é baseada em um computador pessoal com CPU Intel Core™2 Quad 2.8GHz e placa de vídeo NVIDIA GeForce GTX 285, rodando o sistema operacional Linux.

Para apresentar o algoritmo proposto neste trabalho – MHS – foram escolhidas instâncias dos problema de predição de estrutura de proteínas, de otimização estrutural de treliças, além de algumas funções matemáticas, os quais são comumente referenciados na literatura.

O foco dos experimentos foi comparar as implementações em CPU com a implementação do MHS em GPU, realizando medições de tempo de processamento e qualidade de solução, fazendo um comparativo de desempenho, identificando possíveis melhorias que o uso de uma população e a sua implementação em GPU pode trazer para a Busca Harmônica. Para isto, o primeiro problema escolhido para ser analisado foi o dobramento de proteínas AB-2D, por ser um problema de otimização matematicamente complexo e de difícil solução. A instância escolhida foi a sequência de Fibonacci de 21 aminoácidos, a qual necessita de um algoritmo com boas características de exploração do espaço de busca. O segundo problema escolhido, foi o problema de otimização estrutural de 200 barras pelas características implícitas de paralelismo e por apresentar um grande número de restrições, porém, em um espaço de busca relativamente mais simples.

A partir dos resultados experimentais destes problemas, foram realizadas análises estatísticas para identificar os melhores conjuntos de parâmetros de controle do algoritmo, baseadas em teste de normalidade, análise de variância e teste de Scott-Knott.

Como toda meta-heurística, a HS requer um conjunto de parâmetros a ser definido para solução dos problemas. Com a inclusão das características da MHS, um número maior de parâmetros necessita ser definido. Para isto, foram executados experimentos fatoriais, atribuindo diferentes valores aos parâmetros. Para cada problema foram executados experimentos

com a HS rodando em CPU com um conjunto de parâmetros reduzido para serem usados como base de comparação. Já os experimentos com a MHS foram executados com um conjunto bastante superior de experimentos, devido ao grande número de parâmetros, que foram sendo reduzidos conforme a identificação de parâmetros gerais que resultam em soluções de boa qualidade.

Os experimentos e resultados estão organizados em seis subseções, sendo analisados sobre diferentes pontos de vista: Identificação de Parâmetros (seção 4.2), Análise da Qualidade de Solução (seção 4.3), Análise de Convergência (seção 4.4), Análise de Desempenho (seção 4.5), Análise das Estratégias Propostas (seção 4.6) e Considerações Gerais (seção 4.7).

4.2 IDENTIFICAÇÃO DE PARÂMETROS

A identificação do melhor conjunto de parâmetros partiu de uma análise estatística, para identificar um conjunto pequeno de parâmetros a serem utilizados como padrão na MHS.

Os resultados dos experimentos fatoriais foram analisados através de testes de normalidade e análise de variância. Foi identificado o grupo de experimentos com melhor qualidade de solução, através da análise da influência de cada valor da variável e selecionando-se o melhor através do teste de Scott-Knott ao nível de significância de 5%. Com isto, foi identificado um conjunto de parâmetros para a MHS que, na maioria dos casos estudados, apresentam resultados de melhor qualidade.

O objetivo deste procedimento foi reduzir o conjunto de parâmetros, para obter um conjunto geral que poderá posteriormente ser aplicado em outros problemas de *benchmark*. Este conjunto geral pode servir como ponto de partida para o ajuste de parâmetros de controle da MHS.

Para o problema de dobramento de proteínas AB-2D, foi realizado o ajuste de parâmetros através de experimentos fatoriais sobre seis parâmetros, conforme apresentado na Tabela 4. Cada configuração (combinação de parâmetros) foram executadas 60 repetições de cada experimento. A combinação destes parâmetros e os resultados são apresentados no Apêndice D, Tabela 9. Os demais parâmetros foram obtidos empiricamente, sendo eles: tamanho da memória de harmonias, $HMS = 20$ e o *Fret Width*, $FW = 15$. O máximo número de avaliações da função objetivo foi fixado em 5 milhões.

O tamanho da memória de Arranjos Musicais foi selecionado para 32. Este tamanho além de apresentar boas soluções em testes preliminares, foi escolhido para ativar todas as *threads* dos *warps*. Isto permite atingir o maior desempenho possível para a implementação da

MHS – mais detalhes podem ser encontrados na seção 2.1 e na seção 3.4.

Tabela 4 – Parâmetros utilizados para os experimentos fatoriais com o problema de dobramento AB-2D na sequência de 21 aminoácidos

Parâmetro	Descrição	Valores
<i>TS</i>	Tamanho do torneio	{1; 3; 5}
<i>MBL</i>	Máximo comprimento dos blocos construtivos	{1; 3; 5}
<i>UBHR</i>	Taxa de utilização da Harmonia Base	{0; 0,3}
<i>EXP</i>	Número de rodadas sem melhoria para Explosão	{1.000; 10.000; ∞ (SEM EXPLOSÃO)}
<i>HMCR</i>	Taxa de consideração da Memória Harmônica	{80; 90; 99}
<i>PAR</i>	Taxa de ajuste fino	{20; 30}

Fonte: Autoria própria.

A partir desta combinação (*TS*, *MBL*, *UBHR*, *EXP*, *HMCR* e *PAR*) foram realizados 324 experimentos para solucionar o problema do dobramento de proteínas, na sequência de Fibonacci de 21 aminoácidos.

Apresenta-se na Figura 22 um histograma da distribuição das qualidades de solução obtidas nos experimentos com parâmetros pré-fixados. Para verificar se este histograma representa uma distribuição normal foi realizado o teste de normalidade de Kolmogorov-Smirnov. O teste de normalidade indicou que não é possível rejeitar a hipótese de que a distribuição dos resultados médios das qualidades de função obtidas representa com mais de 95% de confiança uma curva normal. Este teste de normalidade e as demais análises estatísticas apresentadas nesta seção são apresentados no Apêndice C.

Aplicou-se na sequência a análise da variância com nível de confiança de 95%, indicando que existem diferenças significativas entre os 324 experimentos, permitindo identificar experimentos de boa qualidade. Na sequência realizou-se o teste de Scott-Knott, com nível de significância de 5%, para separar os resultados em grupos com diferença de resultados estatisticamente significativa. Este teste foi aplicado para os tratamentos dos parâmetros *TS*, *MBL*, *UBHR* e *EXP*. A partir deste teste identificou-se os parâmetros que apresentaram, individualmente, resultados estatisticamente melhores, sendo estes: $TS = 5$, $MBL = \{1; 5\}$, $UBHR = 0,3$ e $EXP = \{1.000; \infty\}$.

A segunda etapa de experimentos fatoriais foi realizada com a instância de 200 barras do problema de otimização da treliça (ver seção 2.4.2), sendo realizados 24 experimentos com as combinações dos parâmetros identificados até o momento (os resultados experimentais deste

problema são apresentados no Apêndice D, na Tabela 10). Assim como na primeira etapa, alguns conjuntos de parâmetros se sobressaíram apresentando qualidade de solução melhor aos demais.

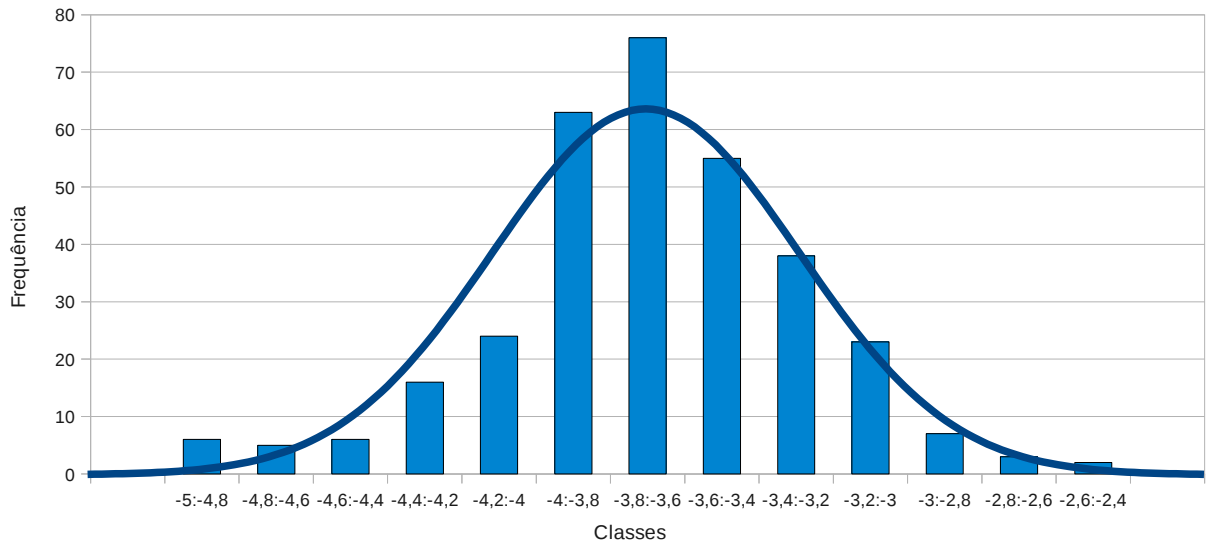


Figura 22 – Histograma da qualidade de solução dos experimentos realizados para o dobramento de proteínas com 21 aminoácidos.

Fonte: Autoria própria.

A aplicação do teste de normalidade Kolmogorov-Smirnov, e posterior análise da variância, ambos com nível de significância 5%, indicou que pelo menos um experimento difere dos demais, sendo um indício de que é possível isolar um grupo de parâmetros. O teste de Scott-Knott a um nível de significância de 5% sobre os parâmetros *MBL* e *EXP*, individualmente, indicou que o uso dos valores de parâmetros $MBL = 5$ e $EXP = \infty$ apresentam soluções estatisticamente melhores. Isto pode ser observado também no gráfico *boxplot* da Figura 23. Os experimentos se dividiram em 4 grupos distintos: o primeiro (experimentos 331 a 336), apresenta os experimentos com parâmetros $MBL = 1$ e $EXP = 1000$; o segundo grupo (experimentos 337 a 342), $MBL = 1$ e $EXP = \infty$; o terceiro grupo (experimentos 343 a 348), $MBL = 5$ e $EXP = 1000$; e o quarto grupo (experimentos 349 a 354), $MBL = 5$ e $EXP = \infty$.

Os parâmetros encontrados a partir da análise realizada foram: $TS = 5$, $MBL = 5$, $UBHR = 0,3$ e $EXP = \infty$. Este conjunto de parâmetros pode ser um potencial conjunto de parâmetros-padrão para o modelo proposto. Não definiu-se valores padrão para *HMCR* e *PAR*, considerando que estes mantenham sua característica de balancear o processo de exploração e intensificação, apesar da influência dos novos parâmetro, sendo eles diferentes para cada tipo e instância de problema.

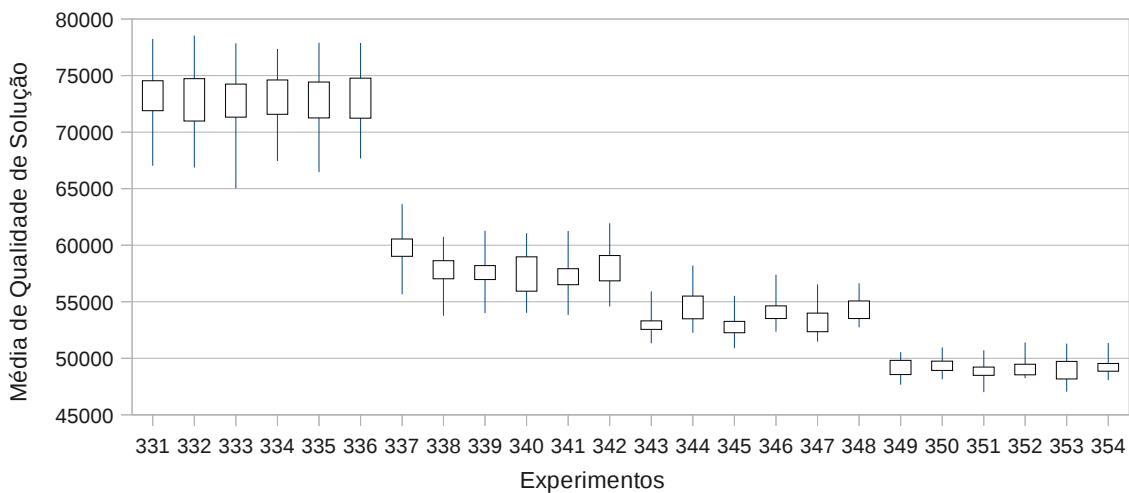


Figura 23 – Gráfico *boxplot* dos experimentos selecionados para o problema de otimização de treliças para a instância de 200 barras.

Fonte: Autoria própria.

4.3 ANÁLISE DA QUALIDADE DE SOLUÇÃO

Nesta seção são analisadas as soluções obtidas para alguns problema, comparando médias e melhores soluções obtidos pela MHS, usando os parâmetros base, com os obtidos com a HS original.

Foi analisada a qualidade de solução obtida pelos seguintes problemas: o problema de dobramento de proteínas AB-2D para as instâncias da sequência de Fibonacci com 13, 21 e 34 aminoácidos; o problema de otimização de treliças para as instâncias de 10 e 200 barras; e para as funções de *benchmark* Griewank, Rosenbrock e Schaffer utilizando dimensões de tamanho 30 e 50.

Para a maioria destes problemas foram realizados seis experimentos fatoriais com a HS, alternando os parâmetros $HMCR = \{80; 90; 99\}$ e $PAR = \{20; 30\}$, exceto para as funções matemáticas nas quais foi considerado $PAR = \{10; 20\}$, sendo os demais parâmetros atribuídos empiricamente, conforme o conjunto de parâmetros apresentados por Lee e Geem (2004). Para a MHS serão considerados seis experimentos utilizando os mesmos valores de $HMCR$ e PAR e o conjunto de parâmetros identificado na seção 4.2.

Uma visão geral dos experimentos realizados para cada instância dos problemas é apresentada na Tabela 5, na qual são mostrados os valores de média e melhor qualidade de solução obtidos utilizando as meta-heurísticas HS e MHS. No Apêndice D são detalhados os resultados experimentais de cada problema, apresentado configuração de parâmetros, tempo médio de processamento, qualidade de solução média e melhor (mínima).

Tabela 5 – Qualidade de solução dos problemas com o uso das meta-heurística HS e MHS

Problema	HS		MHS		Referência da Literatura	
	Média	Melhor	Média	Melhor		
AB-2D(13)	-2,79±0,6	-3,26	-2,11±0,41	-3,13	-3,2245	*
AB-2D(21)	-2,67±0,43	-3,46	-3,92±0,49	-6,09	-5,2881	*
AB-2D(34)	-0,20±2,11	-1,45	-1,02±0,64	-2,41	-8,9749	*
Treliça 10 barras	5073,8±9,25	5063,3	5085,9±81,9	5070,9	5057,88	**
Treliça 200 barras	48740,6±8518	47710,5	48746,4±8866	47015,9	25446,17	***
Griewank(30)	0,0112±0,01	0,0001	0,0457±0,33	0,0107	0	
Griewank(50)	0,0189±0,46	0,0049	0,6774±0,20	0,4904	0	
Rosenbrock(30)	0,20±0,56	0,0012	0,62±1,62	0,0006	0	
Rosenbrock(50)	5,34±5,14	0,004	37,60±36,29	0,013	0	
Schaffer(30)	0,49±0,39	0,39	1,47±1,16	0,54	0	
Schaffer(50)	3,94±1,52	2,66	6,75±2,62	4,79	0	

Fonte: Autoria própria, referências de: * (STILLINGER; HEAD-GORDON, 1995), ** (LEE; GEEM, 2004) e *** (LAMBERTI; PAPPALETTERE, 2003) .

O número de avaliações de cada problema foi idêntico, ou pelo menos próximo, aos resultados de referência ou da literatura.

A MHS apresentou resultados equivalentes em todos os experimentos quando comparado a HS, tanto em média, quanto na melhor qualidade de solução, superando em alguns casos a qualidade de solução obtida pela HS. Porém, em outros experimentos foi superado pela HS. Apenas no experimento do dobramento de proteínas com 21 aminoácidos a melhor solução encontrada pelo algoritmo proposto superou os valores da literatura.

4.4 ANÁLISE DE CONVERGÊNCIA

Nesta seção são analisados os gráficos de curva de convergência para cada problema, comparando a média das soluções melhor conjunto de parâmetros de controle da MHS, com a média das solução do melhor conjunto de parâmetros da HS.

A Figura 24 mostra a curva de convergência da função objetivo para o problema de dobramento de proteínas AB-2D para a instância de 13 aminoácidos. Como pode-se observar, neste experimento, a MHS apresenta uma curva mais suave que a HS. Em geral, deseja-se esta curva mais suave, indicando uma convergência mais lenta, o que possibilita a obtenção de melhores soluções com o prolongamento da execução do algoritmo; entretanto, aumentando o número de avaliações de função. Todavia, a curva de convergência neste caso é muito lenta, não permitindo a obtenção de soluções de boa qualidade, dado o limite de avaliações.

A Figura 25 mostra a curva de convergência da função objetivo para o problema de do-

bramento de proteínas AB-2D para a instância de 21 aminoácidos. A curva de convergência da MHS, neste experimento, apesar de suave, apresenta uma boa qualidade de solução, superando a HS logo no início do processo de otimização. Nota-se que o valor final alcançado pela MHS foi significativamente superior do que o obtido pela HS.

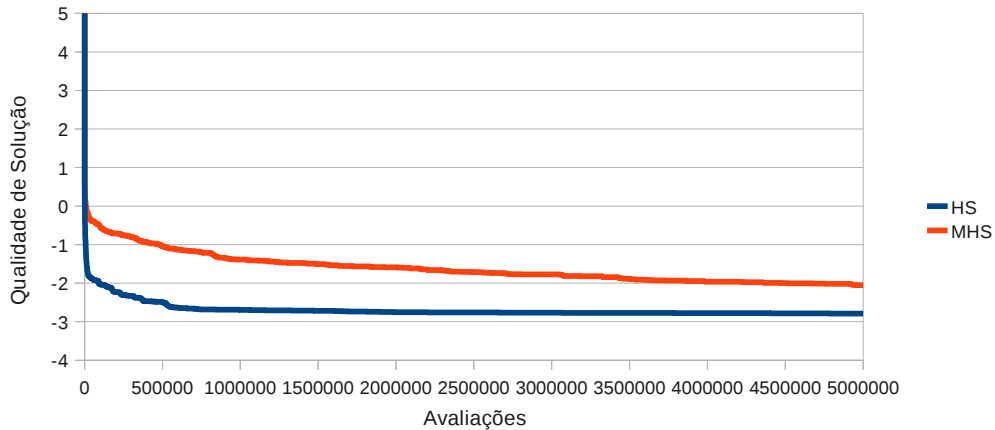


Figura 24 – Curvas de convergência do problema de dobramento de proteínas, 13 aminoácidos
Fonte: Autoria própria.

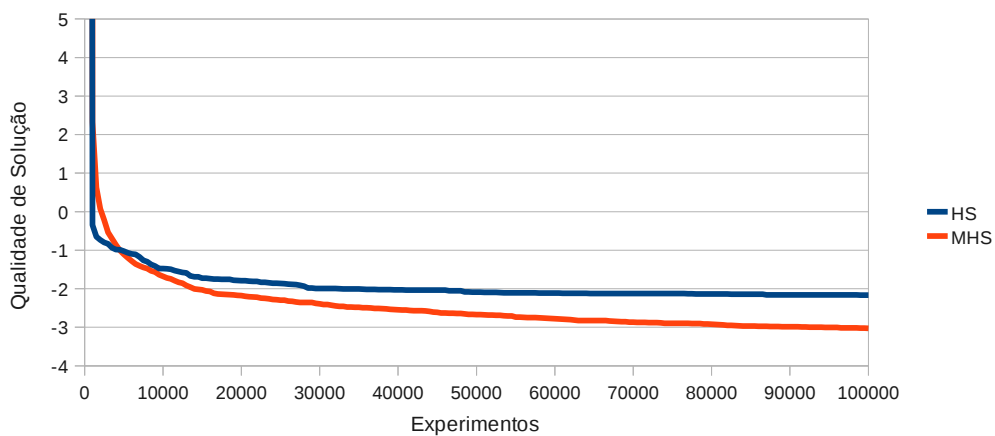


Figura 25 – Curvas de convergência do problema de dobramento de proteínas, 21 aminoácidos
Fonte: Autoria própria.

A Figura 26 mostra a curva de convergência da função objetivo para o problema de dobramento de proteínas AB-2D para a instância de 34 aminoácidos. A HS convergiu rapidamente a um ótimo local, não apresentando melhoria significativa na qualidade de solução durante muitas avaliações de função. Apenas um processo de explosão permitiria a HS, em um evento aleatório, sair deste ótimo local. Já a MHS apresentou uma curva suave, realizando uma exploração de qualidade sobre o espaço de busca, obtendo, assim, uma solução de melhor qualidade. Neste exemplo pode-se observar o quanto o uso da população auxiliou o processo de busca, utilizando soluções potencialmente boas, que na HS haveriam sido desconsideradas.

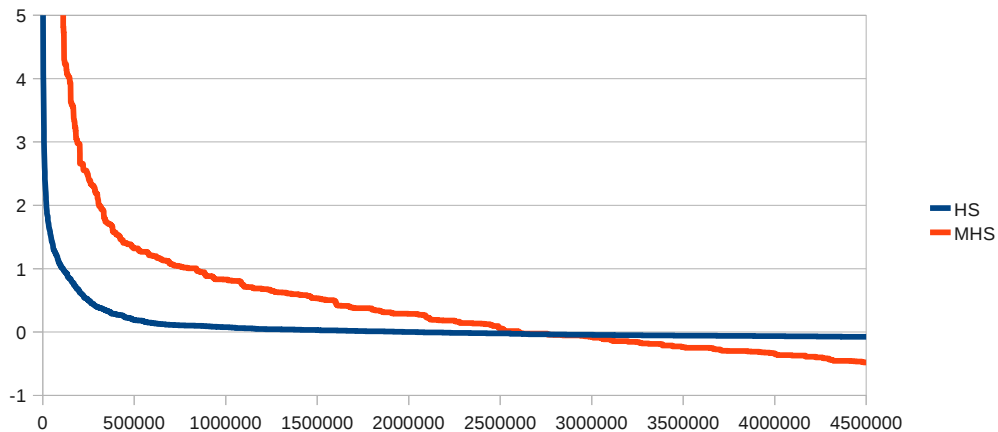


Figura 26 – Curvas de convergência do problema de dobramento de proteínas, 34 aminoácidos
Fonte: Autoria própria.

A Figura 27 mostra a curva de convergência da função objetivo para o problema de otimização estrutural de treliças para a instância de 10 barras. Este problema é de mais fácil resolução que os anteriores, apresentando um número menor de restrições. A MHS apresentou um convergência mais rápida convergindo com aproximadamente em 12.000 avaliações em um máximo local a 0,1% da melhor solução encontrada pela HS. Em média, a HS obteve soluções de qualidade superiores a MHS após 25.000 avaliações.

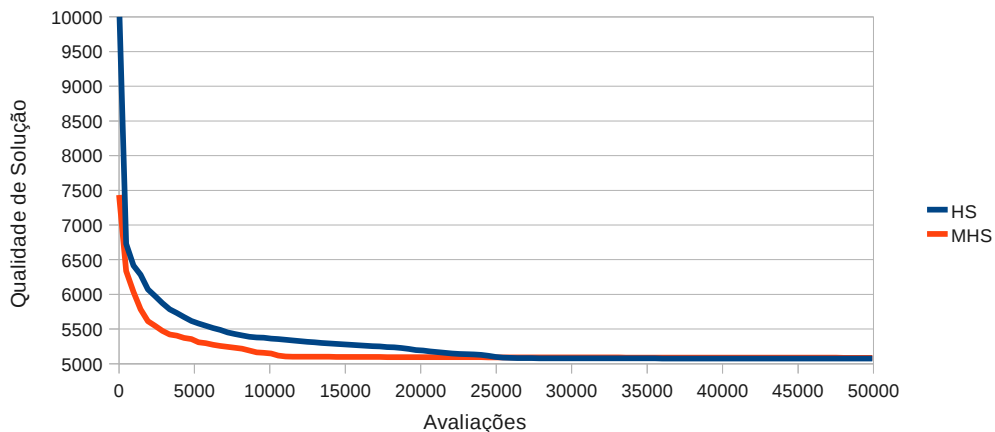


Figura 27 – Curvas de convergência do problema de otimização estrutural de treliças, 10 barras
Fonte: Autoria própria.

A Figura 28 mostra a curva de convergência da função objetivo para o problema de otimização estrutural de treliças para a instância de 200 barras. Como no problema anterior, neste experimento a MHS apresenta uma curva mais suave que a HS, enquanto a HS converge próximo de 25.000 avaliações, a MHS permanece melhorando sua qualidade de solução. Porém, uma solução de boa qualidade não foi encontrada pela MHS com o número de avaliações proposto. Com um número maior de avaliações há a possibilidade de a MHS superar a qualidade de solução da HS.

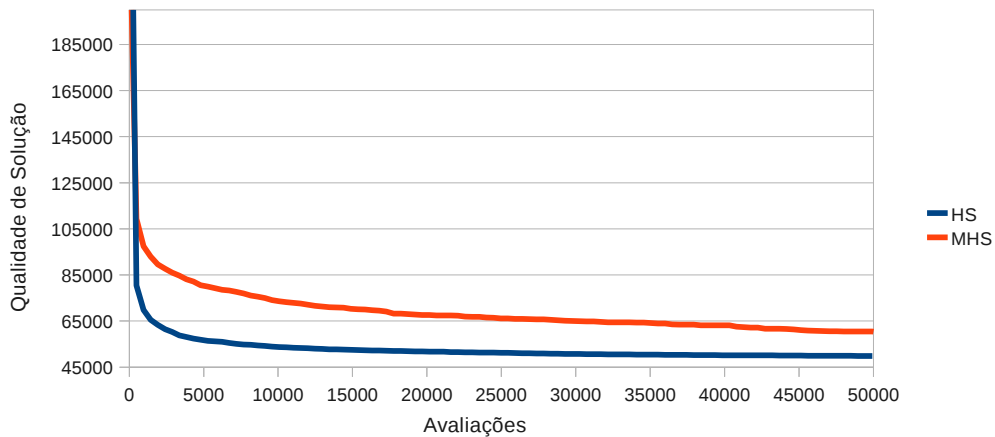


Figura 28 – Curvas de convergência do problema de otimização estrutural de treliças, 200 barras
Fonte: Autoria própria.

A Figura 29 mostra a curva de convergência da função objetivo para a função de Griewank com 30 dimensões. Este, como as funções seguintes, não apresenta restrições explícitas. Sendo assim, apresentam um espaço de busca mais simples e fácil de ser explorado. A convergência da MHS foi abrupta, em comparação da HS, atingindo o valor de 1,1 próximo de 8.000 avaliações. A HS precisou de aproximadamente 60.000 avaliações para superar este valor, apresentando qualidade de solução melhor que a MHS apenas após a avaliação 63.000, enquanto a MHS permanecia estagnado próximo de 1,01.

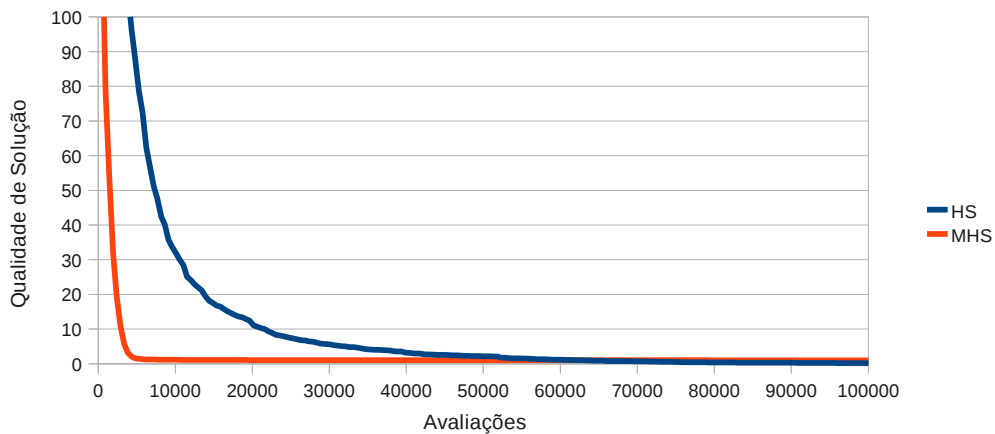


Figura 29 – Curvas de convergência da função de Griewank com 30 dimensões
Fonte: Autoria própria.

A Figura 30 mostra a curva de convergência da função objetivo para a função de Griewank com 50 dimensões. Neste problema, com dimensão maior, a curva de convergência da MHS teve um comportamento mais ameno do que no problema anterior. Porém, a curva apresenta também uma convergência mais rápida do que o HS original. A MHS convergiu antes da avaliação de número 20.000, apresentando valores próximos de 1,3. A HS superou este valor de qualidade de solução apenas em torno da avaliação 50.000, superando a MHS próximo da

avaliação 55.000 com valor 1,12. Isto indica que a MHS teve uma melhora de apenas 0,2 na qualidade de solução em mais de 30.000 avaliações.

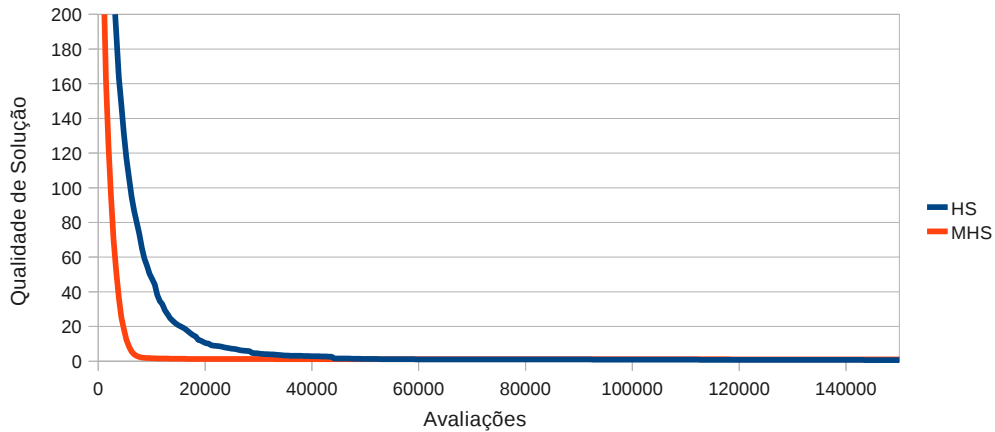


Figura 30 – Curvas de convergência da função de Griewank com 50 dimensões
Fonte: Autoria própria.

A Figura 31 mostra a curva de convergência da função objetivo para a função de Rosenbrock com 30 dimensões. Neste problema, a MHS apresentou uma convergência rápida combinada com uma curva mais suave, e apresentou melhor qualidade de solução que a HS até aproximadamente 900.000 de avaliações. Se tivesse sido considerado um valor de avaliações inferior, a MHS teria superado a HS tanto em qualidade de solução quanto em tempo de convergência.

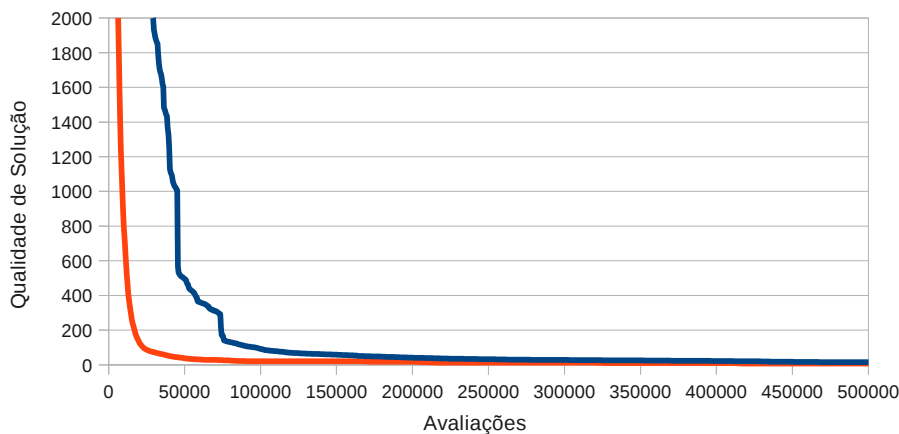


Figura 31 – Curvas de convergência da função de Rosenbrock com 30 dimensões
Fonte: Autoria própria.

A Figura 32 mostra a curva de convergência da função objetivo para a função de Rosenbrock com 50 dimensões. Quando comparado a HS, A MHS apresentou uma convergência rápida, convergindo ao valor de 81,0 próximo da 60.000ª avaliação, e mantendo soluções de melhor qualidade por aproximadamente 382.000 avaliações.

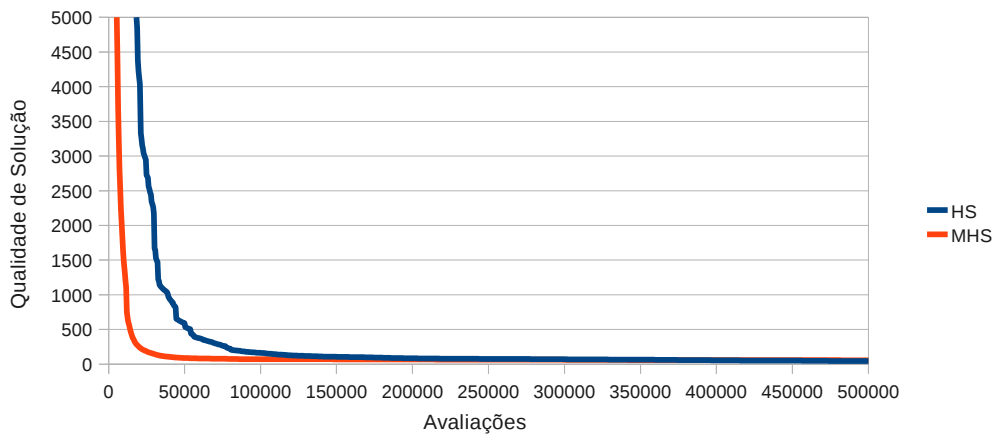


Figura 32 – Curvas de convergência da função de Rosenbrock com 50 dimensões
Fonte: Autoria própria.

A Figura 33 mostra a curva de convergência da função objetivo para a função de Schaffer com 30 dimensões. Para este problema a MHS apresentou uma curva diferente dos problemas matemáticos sem restrição mostrados até o momento. A MHS apresentou uma curva de convergência mais suave e mais lenta, não superando os valores da HS em nenhuma das situações. Porém, após a avaliação de número 200.000, a taxa de melhoria da MHS se mantém superior a HS. Este é um indicativo de que, se forem permitidas mais avaliações, a MHS poderia superar a HS em qualidade de solução.

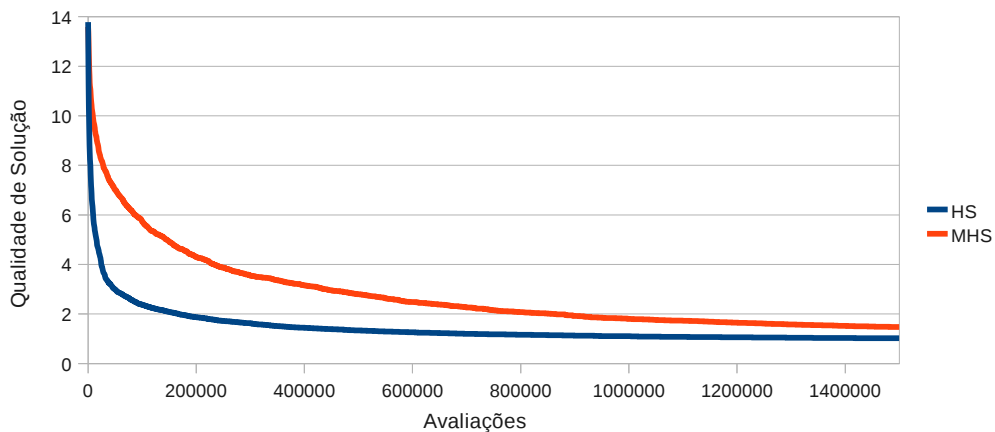


Figura 33 – Curvas de convergência da função de Schaffer com 30 dimensões
Fonte: Autoria própria.

A Figura 34 mostra a curva de convergência da função objetivo para a função de Schaffer com 50 dimensões. A mesma característica de convergência observada com Schaffer 30 dimensões pode ser observada com este problema, porém com maior intensidade.

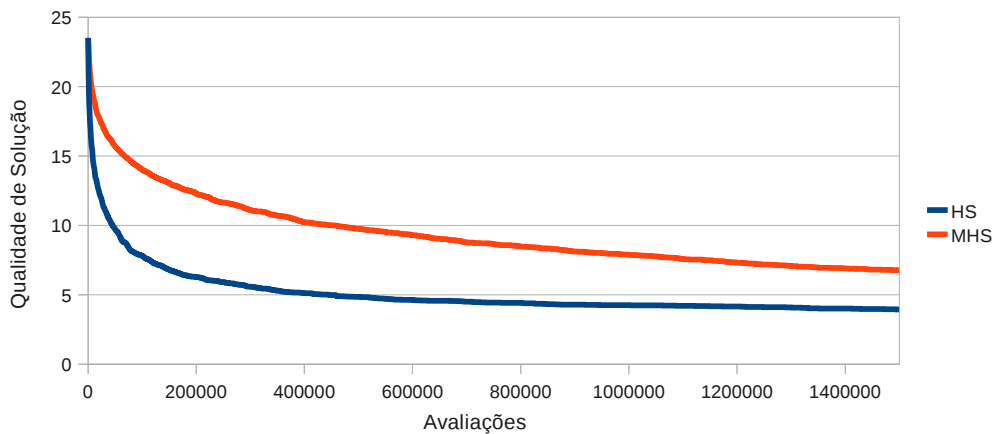


Figura 34 – Curvas de convergência da função de Schaffer com 50 dimensões
Fonte: Autoria própria.

4.5 ANÁLISE DE DESEMPENHO

O objetivo da análise de desempenho é calcular os ganhos de velocidade (*speed-ups*), identificando quantas vezes a MHS foi mais rápida que a HS. A análise de desempenho foi baseada no comparativo do tempo de processamento entre as implementações da MHS executada em CPU com a MHS executada em GPU. A Tabela 6 apresenta um resumo dos tempos médios de processamento para cada problema para as duas implementações.

Tabela 6 – Resumo dos tempos de processamento para cada problema

Problema	Tempo de Processamento (s)		<i>Speed-up</i>
	CPU	GPU	
AB-2D(13)	111,0	17,6	6,3x
AB-2D(21)	276,7	20,2	13,7x
AB-2D(34)	760,8	34,6	21,9x
Treliça 10 barras	1,05	0,43	2,4x
Treliça 200 barras	32,28	4,8	6,7x
Griewank(30)	14,64	2,7	5,4x
Griewank(50)	22,2	3,1	7,1x
Rosenbrock(30)	12,48	3,5	3,5x
Rosenbrock(50)	19,44	4,0	4,8x
Schaffer(30)	8,4	2,1	4,0x
Schaffer(50)	13,32	2,5	5,3x

Fonte: Autoria própria.

Para a implementação proposta a GPU superou a CPU, no quesito tempo de processamento, em todos os experimentos. Foi obtido o maior ganho de velocidade no problema de dobramento de proteínas na sequência de 34 aminoácidos, com o valor de 21,9x.

A Figura 35 mostra os ganhos de velocidade para o dobramento de proteínas. Quanto

maior a dimensão, maior foi o ganho de velocidade obtido. Para efeito de comparação apenas de tempo realizou-se experimentos com uma sequência de Fibonacci de 89 aminoácidos obtendo o ganho de 60,3x (2991,3 segundos em CPU contra 49,6 segundos em GPU). É possível ganhos ainda maiores para instâncias do problema com um número maior de aminoácidos.

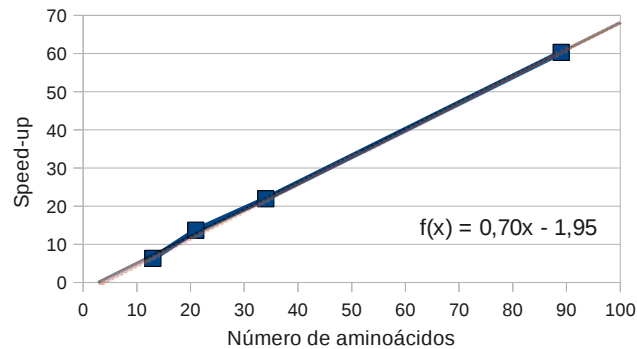


Figura 35 – Speed-ups para diferentes instâncias o dobramento de proteínas.
Fonte: Autoria própria.

4.6 ANÁLISE DAS ESTRATÉGIAS PROPOSTAS

Nesta seção será analisado como cada uma das estratégias propostas influencia de forma independente sobre o processo de otimização. Para tanto são comparadas a qualidade de solução e o desempenho e, também, analisadas das curvas de convergência. Estas análises podem permitir uma melhor escolha dos parâmetros em outros tipos de problemas.

A análise das estratégias foi baseada nos experimentos com o dobramento de proteínas de 21 aminoácidos utilizando como comparativo o experimento 7 apresentado no Apêndice D, Tabela 9. Os parâmetros deste experimento são $HMCR = 80$ e $PAR = 20$.

A primeira estratégia analisada é o método de seleção, apresentada na seção 3.5.1. A estratégia de seleção através do método do torneio estocástico intensifica o processo de busca local, evitando o uso de harmonias com qualidade muito baixa no processo de improvisto. Um parâmetro TS controla o número de harmonias que participam do torneio. Como apresentado anteriormente, foram adotados três valores para o parâmetro TS , sendo eles 1 (seleção aleatória), 3 e 5. Vale salientar que o valor de $TS = 5$ foi selecionado durante a identificação de parâmetros (seção 4.2) como o valor que apresentou melhor qualidade de solução.

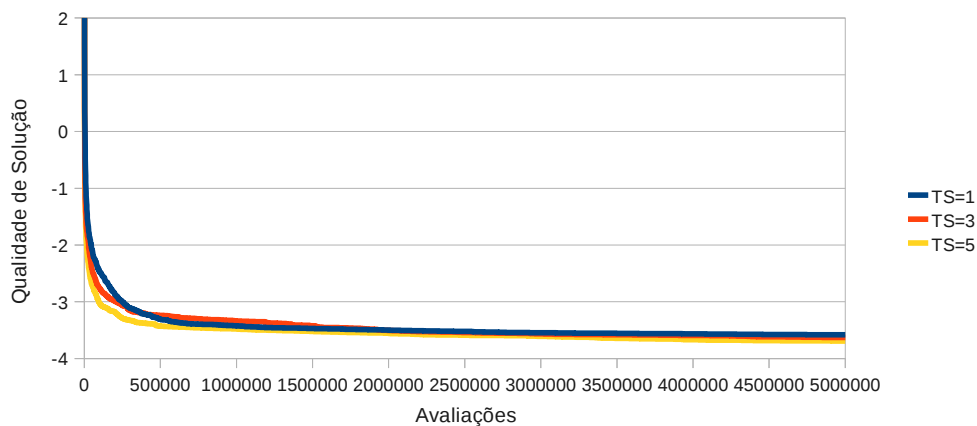
A Tabela 7 apresenta o tempo de processamento e qualidade de solução obtidos nos experimentos. A qualidade de solução melhora à medida que é aumentado o valor de TS , sendo que o valor de $TS = 5$ apresentou a melhor qualidade de solução. Em contrapartida, da mesma forma, quanto maior o tamanho do torneio, maior o tempo de processamento.

Tabela 7 – Qualidade de solução e tempo de processamento para diferentes valores de TS

TS	Média	Melhor	Tempo(s)
1	$-3,58 \pm 0,34$	-4,50	16,96
3	$-3,63 \pm 0,36$	-4,34	18,27
5	$-3,69 \pm 0,54$	-5,06	19,55

Fonte: Autoria própria.

A Figura 36 apresenta as curvas de convergência para os três valores do parâmetro TS . A influência deste parâmetro sobre o processo de otimização é pequena, porém interfere na qualidade de solução de forma importante. A curva gerada pela utilização de $TS = 3$ cruza em dois momentos a curva do $TS = 1$. Isto é, apesar de sua convergência rápida no começo do processo de otimização, ele mantém obtendo soluções de boa qualidade à medida que o processo de otimização evolui. Já a curva gerada pela utilização de $TS = 5$ supera as demais durante todo o processo de otimização, confirmando ser a melhor alternativa de parâmetro.

**Figura 36 – Curvas de convergência para diferentes valores de TS**

Fonte: Autoria própria.

O uso de blocos construtivos maiores, apresentado na seção 3.5.2, tem como objetivo reduzir o número de harmonias que participam do processo de improviso. Isto atribui para a MHS uma característica de intensificação, concentrando esforços em uma determinada região do espaço de busca. Um parâmetro MBL define o tamanho máximo destes blocos construtivos, e para os experimentos foi adotado os valores 1 (sem blocos construtivos), 3 e 5. Através das análises mostradas na seção 4.2 foi identificado o valor 5 como apresentando soluções de melhor qualidade.

A Figura 37 apresenta as curvas de convergência para os três valores do parâmetro MBL .

O aumento do tamanho dos blocos construtivos atenua o processo de otimização, de forma que a curva de evolução se torne um pouco mais suave. Nota-se que as curvas, a partir de

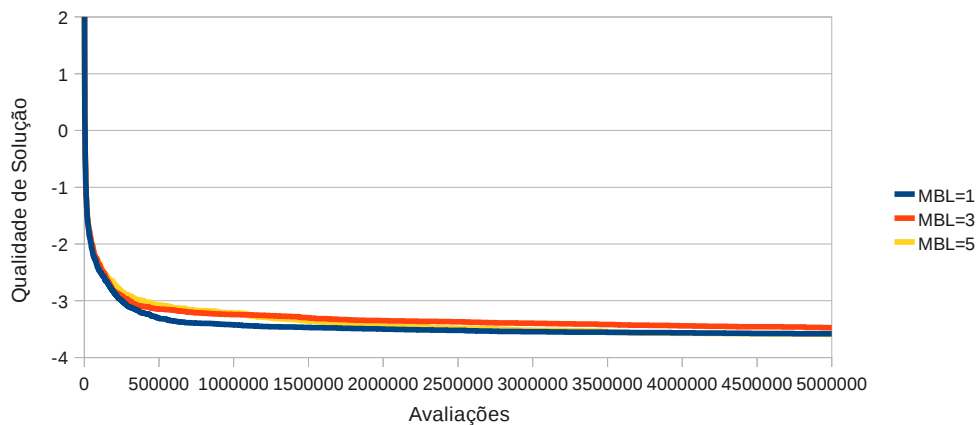


Figura 37 – Curvas de convergência para diferentes valores de *MBL*
Fonte: Autoria própria.

um determinado ponto se sobrepõem. Como $MBL = 5$ apresenta a curva com maior diferencial ela se mostra melhor, podendo obter melhorias na qualidade da solução a longo prazo. Porém, se o número de avaliações é reduzido o uso de valores menores se mostra mais promissor. Como o processo é simples o tempo de processamento é o mesmo independente do valor assumido pelo parâmetro.

A utilização de uma harmonia base, apresentado na seção 3.5.3, tem o mesmo objetivo do uso de blocos construtivos. Porém, este artifício usa uma harmonia como base para improviso, de forma que os valores dela sejam utilizados com maior frequência. O parâmetro *UBHR* define a frequência em que a harmonia base será utilizada, sendo adotados os valores de 0 (sem uso de harmonia base) e 0,3, este último indicando que a harmonia base irá compor aproximadamente 30% das novas harmonias geradas.

A Figura 38 apresenta as curvas de convergência para os dois valores do parâmetro *UBHR*.

A influência desta estratégia não é muito significativa. Porém, pode auxiliar na obtenção de boas qualidades de solução a longo prazo. Neste caso, seus efeitos são notados ao término do processo evolutivo, intensificando a busca em determinadas regiões do espaço e permitindo pequenos ganhos de qualidade de solução. A curva com $UBHR = 30\%$ acompanha a curva na qual não é considerado o uso da harmonia base até aproximadamente 2,67 milhões de avaliações. A partir de então, apresenta solução com qualidade levemente superior. Apesar de pequena esta diferença na qualidade de solução foi suficiente para a identificação de um valor para este parâmetro na seção 4.2, porém quando analisado individualmente não existiu diferença estatística sobre a qualidade de solução nos experimentos com a utilização ou não do mesmo.

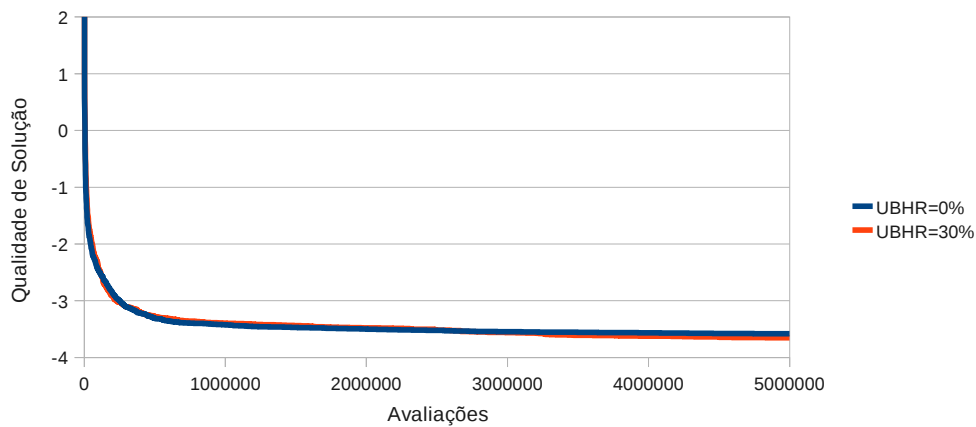


Figura 38 – Curvas de convergência para diferentes valores de $UBHR$
Fonte: Autoria própria.

Também foi proposta uma estratégia de explosão ou dizimação, apresentada na seção 3.5.4. Esta estratégia verifica os critérios de convergência e elimina todas as harmonias da memória harmônica, exceto a de melhor qualidade. Este processo tem como objetivo impedir que a busca fique estagnada, aumentando a variabilidade das harmonias. Um parâmetro EXP é utilizado para verificar há quantas avaliações a melhor harmonia não é substituída. Nos experimentos realizados foram utilizados os valores 1.000, 10.000 e ∞ (sem explosão) para este parâmetro.

As análises realizadas na seção 4.2 mostraram que o uso desta estratégia, da forma como adotada não é benéfica em todas as situações, sendo que o parâmetro $EXP = \infty$ foi adotado como apresentando melhores qualidades de solução, i.e., sem o uso da estratégia.

A Figura 39 apresenta as curvas de convergência para os três valores do parâmetro EXP .

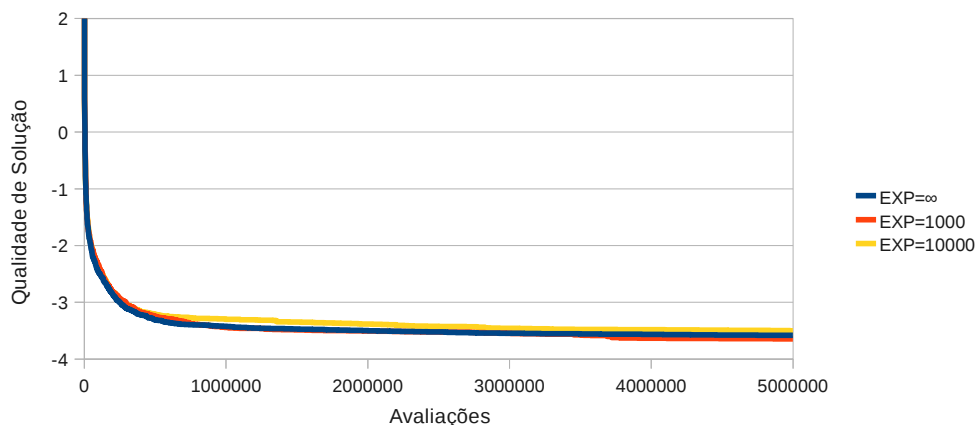


Figura 39 – Curvas de convergência para diferentes valores de EXP
Fonte: Autoria própria.

O uso de explosão torna a curva de convergência mais suave, sendo que valores menores de intervalo após a convergência apresentam melhor qualidade de solução. Observa-se que deixar o processo estagnar muito tempo antes de realizar a explosão pode ser prejudicial ao processo de evolução. Nota-se que tanto o uso de explosão $EXP = 1000$, como o não uso da explosão ($EXP = \infty$), apresentam pior qualidade com um número reduzido de avaliações e soluções de melhor qualidade com número elevado de avaliações. Este fator foi decisivo para definir o parâmetro padrão $EXP = \infty$ durante a identificação de parâmetros (ver seção 4.2) ao realizar-se a análise sobre o experimento de otimização estrutural da treliça de 200 barras, pois o número de avaliações para a treliça é de apenas 50.000. Assim, pode-se afirmar que a utilização de explosão está diretamente ligada ao número de avaliações do problema, sendo que problemas com grande número de avaliações podem fazer o uso de explosão, mas para problemas com número reduzido de avaliações não é recomendado.

A última estratégia a ser analisada é a estratégia de autoadaptação, apresentada na seção 3.5.5. Foram propostas duas estratégias que realizam a autoadaptação dos parâmetros *HMCR* e *PAR*. Na estratégia A0 os valores são atualizados com base na média dos valores da HM. Na estratégia A1 cada harmonia da memória harmônica possui seu próprio conjunto de parâmetros evoluindo independentemente.

A Figura 40 apresenta as curvas de convergência sem o uso de autoadaptação (SA) e com as estratégias A0 e A1.

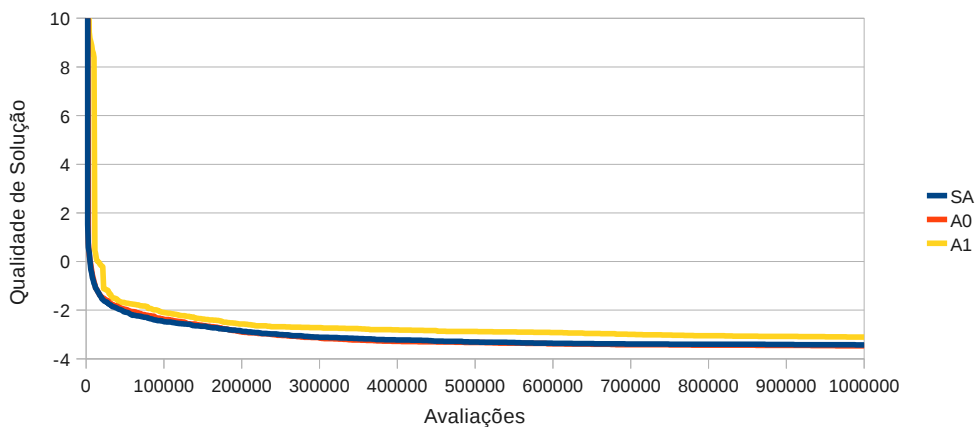


Figura 40 – Curvas de convergência para diferentes estratégias de autoadaptação
Fonte: Autoria própria.

As estratégias de autoadaptação apresentaram qualidade de solução semelhantes às obtidas pelo conjunto de parâmetros padrão, como mostrado na Tabela 8, apresentando um pequeno acréscimo no tempo de processamento. A estratégia A0, que usa as médias dos valores na HM, apresentou uma curva com convergência muito semelhante a curva sem autoadaptação

(SA). O cálculo das médias manteve pouca variação de parâmetros de uma iteração para a outra, sendo assim, faz com que a curva apresente um comportamento regular. Já a estratégia A1 apresentou uma curva com alterações de comportamento, passando de processos de intensificação para processos de exploração e *vice-versa*. Como nesta estratégia cada harmonia atualiza seu conjunto de parâmetros com base no parâmetro selecionado no processo de improviso, as harmonias de melhor qualidade tendem a conduzir a evolução do algoritmo, pois estes possuem maior possibilidade de serem selecionados. Se elas possuírem parâmetros adequados o processo de otimização evolui.

Tabela 8 – Qualidade de solução e tempos de processamento para as estratégias de autoadaptação

Estratégia	Média	Melhor	Tempo(s)
SA	-3,58±0,34	-4,50	16,96
A0	-3,70±0,38	-4,51	17,55
A1	-3,56±0,80	-5,06	27,10

Fonte: Autoria própria.

O uso das estratégias de autoadaptação aumenta a dimensão a ser manipulada pela GPU, tendo um custo computacional adicional para isto. A estratégia A1 apresentou um custo computacional bastante elevado, aumentando em 60% o tempo de processamento, enquanto que a estratégia A0 tem um custo pouco significativo.

Apesar dos experimentos apresentarem curvas de convergência distintas, a análise estatística dos resultados obtidos nesta seção, através do uso individual dos parâmetros, indicaram que o uso individual dos parâmetros não apresentou diferença significativa, quando comparado o resultado final ao resultado do experimento sem o uso das estratégias propostas. Isto pode ser observado nos testes de Scott-Knott realizados apresentados no Apêndice C.

4.7 CONSIDERAÇÕES GERAIS

Este capítulo apresentou os experimentos realizados e análises dos resultados obtidos.

Identificou-se um conjunto de parâmetros-padrão que apresentou bons resultados para a resolução de diferentes problemas. A escolha de cada parâmetro foi analisada individualmente através de gráficos da curva de convergência, sendo que, mesmo combinados, eles apresentam soluções de melhor qualidade.

Como observado, o modelo proposto (MHS) apresenta características de comportamento bastante interessantes, obtendo soluções de qualidade competitivo a HS. A MHS apresenta convergência mais rápida para os problemas simples, com boa qualidade de solução e um número reduzido de avaliações. Para problema complexos e com grande número de restrições

o comportamento se inverte, a MHS apresenta uma convergência mais suave que a HS, que sugere, teoricamente, a possibilidade de se obter melhores soluções dado um pouco mais de tempo de processamento.

O sucesso da MHS se dá justamente por causa da manutenção de diversidade decorrente da presença da população, o que permite manter soluções igualmente boas e otimizá-las a cada iteração, num processo de exploração do espaço de busca. A combinação das estratégias adicionais possibilita contrabalancear o processo exploratório com a intensificação do processo de otimização.

Ao final da execução dos experimentos, verificou-se que o uso de GPU na execução do MHS apresentou uma melhora de desempenho significativa em todos os experimentos, obtendo ganhos de velocidade de até 60,3x, que podem ser maiores para problemas mais complexos. Isto permitiria, caso necessário, executar o algoritmo por um tempo maior para encontrar soluções de melhor qualidade no mesmo tempo de execução.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve como objetivo desenvolver e analisar um novo algoritmo de otimização utilizando GPU aplicado a diversos problemas. Foi realizada uma adaptação do algoritmo HS, com a implementação de uma população auxiliar. A presença desta população permite a execução paralela de várias avaliações de função com o uso da arquitetura CUDA. O uso desta arquitetura em GPU permitiu atingir ganhos de desempenho através da execução paralela deste modelo.

A meta-heurística proposta faz o uso de uma população de indivíduos temporários que são substituídos a cada iteração do algoritmo. A presença destes novos indivíduos permite uma melhor exploração do espaço de busca com o processamento de mais de um indivíduo por iteração. Este paralelismo é o fator preponderante para a redução do tempo de processamento, quando em arquiteturas paralelas.

Os testes realizados compararam o modelo proposto (MHS) ao algoritmo que serviu como inspiração (HS) através da resolução de problemas de dobramento de proteínas, problemas de otimização estrutural e problemas matemáticos, os quais foram adotados como casos de estudo.

O ajuste de parâmetros foi baseado na execução de testes fatoriais – com os problemas de dobramento de proteínas com 21 aminoácidos e otimização estrutural da treliça de 200 barras – e em análises estatísticas sobre os resultados. Isto permitiu a identificação de um conjunto de parâmetros padrão que apresentou soluções de qualidade equivalente à HS nos experimentos realizados, podendo efetivamente servir como sugestão de parâmetros iniciais para a resolução de outros problemas.

Os parâmetros identificados foram utilizados, então, para o processamento de outros problemas de *benchmark* e os resultados foram analisados sobre diferentes pontos de vista, indicando que, de maneira geral, foram obtidos resultados com qualidades de solução competitivas com a HS e que apresentam ganhos de velocidade consideráveis, podendo superar um ganho de 60x. Permitindo obter uma redução significativa no tempo de processamento.

Apesar das soluções serem competitivas com a HS, o uso do conjunto de parâmetros padrão não garante a obtenção de soluções melhores, como pode ser observado na comparação com os valores encontrados por outros trabalhos.

Foram realizados experimentos utilizando os parâmetros identificados com o problema de dobramento de proteínas com 21 aminoácidos, desta vez comparando a MHS sem nenhuma das estratégias. A utilização individualmente de cada estratégia reafirmou a escolha dos parâmetros identificados. Os experimentos utilizando a autoadaptação apresentaram soluções equivalentes ao uso dos parâmetros pré-fixados, simplificando o processo de configuração do algoritmo.

A principal contribuição deste trabalho é a apresentação de uma nova abordagem que pode ser dada a meta-heurísticas elitistas permitindo ganhos de velocidade quando implementadas em arquiteturas paralelas, além da incorporação de recursos adicionais inspirados em outras meta-heurísticas que permitem melhorar as qualidades de solução que podem ser obtidas.

Para trabalhos futuros, alguns aspectos do algoritmo MHS podem ser ainda melhorados, tanto em desempenho computacional, como em qualidade de resultados. Novas análises sobre a influência de seus parâmetros podem ser realizadas, como, por exemplo, a influência do tamanho da população de arranjos musicais (*PS*) e a combinação de seus parâmetros em diversos tipos de problemas. Pretende-se também implementar e estudar novas estratégias, como a execução paralela de vários controles de fluxo utilizando arquiteturas CUDA Fermi, partindo desta, a implementação de nichos e estratégias de interação entre eles.

Acredita-se que, em algumas situações, o aumento do tamanho da população em problemas mais simples possa tornar a curva de convergência mais suave, convergindo com um número maior de avaliações, podendo, assim, obter soluções de melhor qualidade. Também o aumento da população permitiria a execução de um número ainda maior de avaliações simultâneas aumentando os ganhos de velocidade, contrabalanceando o número adicional de avaliações – fato observado em experimentos com população de tamanho 64, não apresentados neste trabalho, confirma estas afirmações, motivando seu estudo em trabalhos futuros.

Finalizando, este trabalho mostrou que o uso da GPU para resolução de problemas através de meta-heurísticas populacionais é uma alternativa viável, que pode reduzir consideravelmente o custo computacional. Fazendo-se o uso da abordagem adequada, é possível usar a GPU como um artifício importante para reduzir o tempo de processamento global, também em outras meta-heurísticas baseadas em população, como AG e PSO, computando o algoritmo e as avaliações de função de muitos indivíduos paralelamente.

REFERÊNCIAS

- AMABIS, J.; MARTHO, G. **Fundamentos da Biologia Moderna**. São Paulo: Editora Moderna, 1990.
- APPLE COMPUTER INC. **Apple Previews Mac OS X Snow Leopard to Developers**. 2008. Press Release. Disponível em: <<http://www.apple.com/pr/library/2008/06/09snowleopard.html>>.
- ARORA, R.; TULSHYAN, R.; DEB, K. Parallelization of binary and real-coded genetic algorithms on CUDA. In: **Proceedings of the 2010 IEEE Congress on Evolutionary Computation**. Barcelona, Espanha: [s.n.], 2010. p. 1–8.
- ATKINS, J.; HART, W. On the intractability of protein folding with a finite alphabet. **Algorithmica**, v. 25, n. 2-3, p. 279–294, 1999.
- AYVAZ, M. Simultaneous determination of aquifer parameters and zone structures with fuzzy c-means clustering and meta-heuristic harmony search algorithm. **Advances in Water Resources**, v. 30, n. 11, p. 2326–2338, 2007.
- BACHMANN, M.; ARKIN, H.; JANKE, W. Multicanonical study of coarse-grained off-lattice models for folding heteropolymers. **Physical Review E**, v. 71, n. 3, p. 031906, 2005.
- BENÍTEZ, C.; LOPES, H. Hierarchical parallel genetic algorithm applied to the three-dimensional hp side-chain protein folding problem. In: **Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics**. [S.l.]: IEEE Computer Society, 2010. p. 2669–2676.
- BERGER, B.; LEIGHTON, F. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. **Journal of Computational Biology**, v. 5, n. 1, p. 27–40, 1998.
- BERKE, L.; KHOT, N. S. Use of optimality criteria methods for large scale system. **AGARD Lectures Series**, n. 70, p. 1–29, 1974.
- BOUZY, G.; ABEL, J. F. A two-step procedure for discrete minimization of truss weight. In: **Structural and Multidisciplinary Optimization**. Heidelberg, Germany: Springer-Verlag, 1995. v. 9, p. 128–131.
- BOŻEJKO, W.; SMUTNICKI, C.; UCHROŃSKI, M. Parallel calculating of the goal function in metaheuristics using GPU. In: **Proceedings of the 9th International Conference on Computational Science: Part I**. Berlin, Heidelberg: Springer-Verlag, 2009. p. 1014–1023.
- BRANDEN, C.; TOOZE, J. **Introduction to Protein Folding**. New York: Garland Publishing, 1999.
- BUCK, I. et al. Brook for GPUs: stream computing on graphics hardware. **ACM Transactions on Graphics**, v. 23, n. 3, p. 777–786, 2004.

CAMP, C.; PEZESHK, S.; CAO, G. Optimized design of two-dimensional structures using a genetic algorithm. **Journal of Structural Engineering**, v. 124, n. 5, p. 551–559, 1998.

CASTRO, L. N. de; TIMMIS, J. An artificial immune network for multimodal optimisation. In: **Proceedings of the 2002 IEEE World Congress on Computational Intelligence**. Honolulu, Hawaii, USA: IEEE Press, 2002. p. 699–704.

CHEN, X. et al. An improved particle swarm optimization for protein folding prediction. **International Journal of Information Engineering and Electronic Business (IJIEEB)**, v. 3, n. 1, p. 1–8, 2011.

CHO, H.; OLIVERA, F.; GUIKEMA, S. A derivation of the number of minima of the Griewank function. **Applied Mathematics and Computation**, v. 204, n. 2, p. 694–701, 2008.

COELLO, C. A. C.; CHRISTIANSEN, A. D. Multiobjective optimization of trusses using genetic algorithms. **Computers and Structures**, v. 75, n. 6, p. 647–660, May 2000.

CRESCENZI, P. et al. On the complexity of protein folding. **Journal of Computational Biology**, v. 5, p. 423–446, 1998.

DARWIN, C. **On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life**. London: John Murray, 1859. 250 p.

DIGALAKIS, J. G.; MARGARITIS, K. G. An experimental study of benchmarking functions for evolutionary algorithms. **International Journal of Computer Mathematics**, v. 79, n. 4, p. 403–416, 2002.

DILL, K. Polymer principles and protein folding. **Protein Science**, v. 8, n. 6, p. 1166–1180, 1999.

DILL, K. et al. Principles of protein folding - a perspective from simple exact models. **Protein Science**, v. 4, n. 4, p. 561–602, 1995.

DOBSON, C. Protein misfolding, evolution and disease. **Trends in Biochemical Sciences**, v. 24, n. 9, p. 329–332, 1999.

FESANGHARY, M. et al. Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. **Computer Methods in Applied Mechanics and Engineering**, v. 197, n. 33-40, p. 3080–3091, 2008.

FOGEL, L. J. **On the Organization of Intellect**. Tese (Doutorado) — Department of Electrical Engineering, University of California, Los Angeles, 1964.

FOK, K.; WONG, T.; WONG, M. Evolutionary computing on consumer graphics hardware. **IEEE Intelligent systems**, Piscataway, NJ, USA, v. 22, n. 2, p. 69–78, March 2007.

GARLAND, M. et al. Parallel computing experiences with CUDA. **IEEE Micro**, v. 28, p. 13–27, 2008.

GEEM, Z. Global optimization using harmony search: theoretical foundations and applications. **Foundations of Computational Intelligence**, v. 3, p. 57–73, 2009.

- GEEM, Z. W. Harmony search algorithm for solving sudoku. In: **Proceedings of the 5th International Conference on Knowledge-Based Intelligent Information and Engineering Systems**. Berlin, Heidelberg: Springer-Verlag, 2007. p. 371–378.
- GEEM, Z. W. State-of-the-art in the structure of harmony search algorithm. In: GEEM, Z. W. (Ed.). **Recent Advances In Harmony Search Algorithm**. Berlin, Heidelberg: Springer, 2010, (Studies in Computational Intelligence, v. 270). p. 1–10.
- GEEM, Z. W.; KIM, J.-H.; LOGANATHAN, G. V. A new heuristic optimization algorithm: harmony search. **Simulation**, v. 76, n. 2, p. 60–68, 2001.
- GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading, USA: Addison-Wesley, 1989.
- HAFTKA, R.; GÜRDAL, Z. **Elements of Structural Optimization**. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1992.
- HAUG, E. J.; ARORA, J. S. Applied optimum design. In: **Mechanical and Structural Systems**. New York, N.Y: John Wiley & Sons Inc., 1979.
- HENEINE, I. F. **Biofísica Básica**. 1^a. ed. São Paulo: Atheneu, 1984.
- HOLLAND, J. **Adaptation in Natural and Artificial Systems**. Cambridge, USA: University of Michigan Press, 1975.
- HSU, H.; MEHRA, V.; GRASSBERGER, P. Structure optimization in an off-lattice protein model. **Physical Review E**, v. 68, n. 3, p. 037703, 2003.
- HUNTER, L. **Artificial Intelligence and Molecular Biology**. 1st. ed. Boston, USA: AAAI Press, 1993.
- KALEGARI, D. H. **Algoritmo de Evolução Diferencial Paralelo Aplicado ao Problema da Predição da Estrutura de Proteínas Utilizando o Modelo AB em 2D e 3D**. 126 p. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná – UTFPR, Curitiba, 2010.
- KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: **Proceedings of the 1995 IEEE International Conference on Neural Networks**. Piscataway, USA: IEEE Press, 1995. p. 1942–1948.
- KHRONOS OpenCL WORKING GROUP. **The OpenCL Specification (Version 1.1)**. [S.l.], 2010. Available from <http://www.khronos.org/opencl>.
- KIRK, D.; HWU, W. Applied parallel programming, chapter 4 - CUDA memories. Draft. 2008.
- KIRK, D.; HWU, W. Lectures 8: threading and memory hardware in G80. Draft. 2009.
- KIRK, D.; HWU, W. Lectures 9: memory hardware in G80. Draft. 2009.
- KIRSCH, U. **Structural Optimization: Fundamentals and Applications**. Heidelberg, Germany: Springer-Verlag, 1993.
- KITA, H. A comparison study of self-adaptation in evolution strategies and real-coded genetic algorithms. **Evolutionary Computation**, MIT Press, v. 9, n. 2, p. 223–241, 2001.

- KOMATITSCH, D. et al. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. **Journal of Computational Physics**, v. 229, n. 20, p. 7672–7714, 2010.
- KOOMEY, J. G. et al. Implications of historical trends in the electrical efficiency of computing. **IEEE Annals of the History of Computing**, v. 33, n. 3, p. 46–54, jul./set. 2011.
- LAMBERTI, L. An efficient simulated annealing algorithm for design optimization of truss structures. **Computers & Structures**, v. 86, n. 19-20, p. 1936–1953, 2008.
- LAMBERTI, L.; PAPPALETERE, C. Move limits definition in structural optimization with sequential linear programming. Part II: Numerical examples. **Computers & structures**, v. 81, n. 4, p. 215–238, 2003.
- LEE, K. S.; GEEM, Z. W. A new structural optimization method based on the harmony search algorithm. **Computers & Structures**, v. 82, n. 9-10, p. 781–798, 2004.
- LEVINE, D. M. et al. **Estatística - Teoria e Aplicações Usando o Microsoft Excel em Português**. 5. ed. São Paulo: LTC (Grupo GEN), 2008.
- LEVINTHAL, C. Are there pathways for protein folding? **Journal de Chimie Physique**, v. 65, p. 44–45, 1968.
- LI, H.; TANG, C.; WINGREEN, N. Nature of driving force for protein folding: a result from analyzing the statistical potential. **Physical Review Letters**, v. 79, n. 4, p. 765–768, 1997.
- LODISH, H. et al. **Molecular Cell Biology**. 4th. ed. New York, NY, USA: Freeman, 2000.
- LOPES, H. Evolutionary algorithms for the protein folding problem: a review and current trends. In: SMOLINSKI, T.; MILANOVA, M.; HASSANIEN, A.-E. (Ed.). **Computational Intelligence in Biomedicine and Bioinformatics**. Heidelberg, Germany: Springer-Verlag, 2008. v. I, p. 297–315.
- MAHDAVI, M.; FESANGHARY, M.; DAMANGIR, E. An improved harmony search algorithm for solving optimization problems. **Applied Mathematics and Computation**, v. 188, n. 2, p. 1567–1579, 2007.
- MOGNON, V. R. **Algoritmos Genéticos Aplicados na Otimização de Antenas**. 85 p. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica com ênfase em Telecomunicações, Universidade Federal do Paraná – UFPR, Curitiba, 2004.
- MOHANTY, S. P. GPU-CPU multi-core for real-time signal processing. In: **Proceedings of the 27th IEEE International Conference on Consumer Electronics**. Los Alamitos, USA: IEEE Computer Society, 2009. p. 55–56.
- MOORE, G. E. Cramping more components onto integrated circuits. **Electronics Magazine**, v. 38, n. 8, p. 114–117, abr. 1965.
- MOORKAMP, M. et al. Massively parallel forward modeling of scalar and tensor gravimetry data. **Computers & Geosciences**, v. 36, n. 5, p. 680–686, 2010.
- NGO, J. T.; MARKS, J.; KARPLUS, M. Computational complexity, protein structure prediction, and the levinthal paradox. In: **Computational Complexity Protein Structure Prediction and the Levinthal Paradox**. [S.l.]: Birkhauser, 1994. p. 433–506.

- NVIDIA Corporation. **CUDA for GPU Computing**. feb 2007.
- NVIDIA CUDA Team. **CUDA Programming Model Overview**. 2008a.
- NVIDIA CUDA Team. **GPU Computing**. 2008b.
- NVIDIA CUDA Team. **NVIDIA Compute PTX: parallel thread execution, ISA version 1.4**. jun 2009.
- NVIDIA CUDA Team. **CUDA Programming Guide Version 3.0**. feb 2010.
- OLIVEIRA, P. R. **Introdução a Programação Não-Linear**. Belo Horizonte: UFMG, 1989.
- OWENS, J. D. et al. A survey of general-purpose computation on graphics hardware. In: **Proceedings of the Eurographics 2005, State of the Art Reports**. Dublin, Ireland: Eurographics Association, 2005. p. 21–51.
- PARHAMI, B. **Introduction to Parallel Processing: Algorithms and Structures**. New York: Kluwer Academic, 2002.
- PEDERSEN, C. **Algorithms in Computational Biology**. Tese (Doutorado) — Department of Computer Science, University of Aarhus, Denmark, 2000.
- RAMALHO, M. A. P.; FERREIRA, D. F.; OLIVEIRA, A. C. de. **A experimentação em genética e melhoramento de plantas**. 2. ed. [S.l.]: UFLA, 2005.
- RECHENBERG, I. **Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution**. Stuttgart: Friedrich Frommann Verlag · Günter Holzboog, 1973. (Problemata).
- SAKA; M.P. Optimum Geometry Design of Geodesic Domes Using Harmony Search Algorithm. **Advances in Structural Engineering**, v. 10, n. 6, p. 595–606, dez. 2007.
- SANTOS, C. dos. **Novas alternativas de testes de agrupamento avaliadas por meio de simulação Monte Carlo**. [S.l.]: Universidade Federal de Lavras, 2000.
- SCALABRIN, M. H.; PARPINELLI, R. S.; LOPES, H. S. Paralelização do algoritmo harmony search utilizando unidade de processamento gráfico. **Proceedings of the CILAMCE 2010 – XXXI Iberian-Latin-American Congress on Computational Methods**, Asociación Argentina de Mecánica Computacional, Buenos Aires, Argentina, v. 29, p. 7109–7121, 2010.
- SCANZIO, S. et al. Parallel implementation of artificial neural network training for speech recognition. **Pattern Recognition Letters**, v. 31, n. 11, p. 1302–1309, 2010.
- SCOTT, A. J.; KNOTT, M. A. A cluster analysis method for grouping means in the analysis of variance. In: **Biometrics**. [S.l.]: Raleigh, 1974. v. 30, n. 3, p. 507–512.
- SHAMS, R. et al. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. **Computer Methods and Programs in Biomedicine**, v. 99, n. 2, p. 133–146, 2010.
- STILLINGER, F.; HEAD-GORDON, T. Collective aspects of protein folding illustrated by a toy model. **Physical Review E**, v. 52, n. 3, p. 2872–2877, 1995.

- SUNARSO, A.; TSUJI, T.; CHONO, S. GPU-accelerated molecular dynamics simulation for study of liquid crystalline flows. **Journal of Computational Physics**, v. 229, n. 15, p. 5486–5497, 2010.
- TAN, Y.; ZHOU, Y. Parallel particle swarm optimization algorithm based on graphic processing units. In: HIOT, L. M. et al. (Ed.). **Handbook of Swarm Intelligence**. Berlin, Heidelberg: Springer-Verlag, 2010, (Adaptation, Learning, and Optimization, v. 8). p. 133–154.
- TANG, C. Simple models of the protein folding problem. **Physica A: Statistical Mechanics and its Applications**, v. 288, n. 1, p. 31–48, 2000.
- THOMASSON, W. A. B. Unraveling the mystery of protein folding. **Breakthroughs in Bioscience**, FASEB Office of Public Affairs, 2001.
- VASEBI, A.; FESANGHARY, M.; BATHAEE, S. Combined heat and power economic dispatch by harmony search algorithm. **International Journal of Electrical Power & Energy Systems**, v. 29, n. 10, p. 713–719, 2007.
- VENKAYYA, V. B. Design of optimum structures. **Computers and Structures**, v. 1, n. 1, p. 265–309, 1971.
- VENKAYYA, V. B.; KHOT, N. S.; REDDY, V. S. Optimization of structures based on the study of energy distribution. In: **Proceedings of the Second Conference on Matrix Methods in Structural Mechanics**. Ohio, USA: [s.n.], 1968. v. 1, n. 1, p. 111–155.
- WANG, Q. **A Study of Alternative Formulations for Optimization of Structural and Mechanical Systems Subjected to Static and Dynamic Loads**. Tese (Doutorado) — The University of Iowa, Iowa, 2006.
- WONG, M.; WONG, T. Parallel hybrid genetic algorithms on consumer-level graphics hardware. In: IEEE. **Proceedings of the 2006 IEEE Congress on Evolutionary Computation**. [S.l.], 2006. p. 2973–2980.
- WONG, M.; WONG, T.; FOK, K. Parallel evolutionary algorithms on graphics processing unit. In: **Proceedings of the 2005 IEEE Congress on Evolutionary Computation**. [S.l.: s.n.], 2005. v. 3, p. 2286–2293.
- YILMAZ, A.; WEBER, G. Why you should consider nature-inspired optimization methods in financial mathematics. In: **Nonlinear and Complex Dynamics: Applications in Physical, Biological, and Financial Systems**. Heidelberg, Germany: Springer, 2011. p. 241–255.
- YOSHIMI, M. et al. An implementation and evaluation of CUDA-based GPGPU framework by genetic algorithms. **International Journal of Computer Science and Network Security**, v. 10, n. 12, p. 29, 2010.
- YU, Q.; CHEN, C.; PAN, Z. Parallel genetic algorithms on programmable graphics hardware. In: WANG, L.; CHEN, K.; ONG, Y. S. (Ed.). **Advances in Natural Computation**. Berlin, Heidelberg: Springer-Verlag, 2005, (Lecture Notes in Computer Science, v. 3612). p. 1051–1059.
- ZHI-PENG, L.; WEN-QI, H.; HE, S. Quasi-physical algorithm for protein folding in an off-lattice model. **Communications in Theoretical Physics**, v. 47, p. 181, 2007.

ZHU, H.; XIAO, H.; GU, J. Parallelism of clonal selection for PSP on CUDA. In: **Proceedings of Third International Conference on Intelligent Networks and Intelligent Systems**. Washington, DC, USA: IEEE Computer Society, 2010. (ICINIS '10), p. 467–470.

ZHU, W. A study of parallel evolution strategy: pattern search on a GPU computing platform. In: ACM. **Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation**. [S.l.], 2009. p. 765–772.

APÊNDICE A – IMPLEMENTAÇÃO USANDO CUDA

Tirar proveito da arquitetura CUDA não é uma tarefa trivial. Primeiramente, é preciso entender como funciona a estrutura da arquitetura, de modo a organizar a aplicação para que possa ser desenvolvida utilizando a tecnologia. Aqui serão apresentados alguns exemplos ilustrativos de código: a conversão de um código C para CUDA; e o uso de memórias em CUDA.

Um princípio bastante prático para desenvolvedores que estão começando a utilizar esta tecnologia é converter laços de tamanho conhecido que atuam sobre variáveis interdependentes em um nível de paralelismo. Um exemplo comumente utilizado na literatura para representar isto é a conversão do Algoritmo 5, em linguagem C, para o Algoritmo 6, em linguagem CUDA. Este algoritmo executa um incremento simples em todas as posições de um vetor. Aqui, o critério de parada do laço (`for (...; idx < N; ...)`) é transformado em uma estrutura de decisão (`if (idx < N)`), servindo como limitador do tamanho do vetor. O número de blocos e *threads* é definido em uma estrutura `dim3`, a qual armazena os valores das dimensões *x*, *y* e *z*. Quando apenas uma dimensão é necessária (*x*) pode-se informar diretamente o valor, não fazendo-se necessário o uso da estrutura `dim3`.

A execução de cada iteração do laço fica sob a responsabilidade de cada *thread* independente. Para isto, cada *thread* calcula seu índice que referencia a posição a ser manipulada no vetor (`int idx = blockIdx.x * blockDim.x + threadIdx.x;`). Para que o número de *threads* seja compatível com o número de elementos do vetor, é definido quantas *threads* para cada bloco (`blocksize`), e em seguida calculado a quantidade de blocos necessária para a manipulação desse vetor. Vale lembrar que para conjuntos pequenos a distribuição em apenas uma dimensão de *threads* poderia ser suficiente.

O Algoritmo 7, mostra o uso da memória global da GPU. As variáveis são alocadas na memória global através do comando `cudaMalloc`. O primeiro argumento é o ponteiro que armazena a posição da memória global, e o segundo argumento é o tamanho do vetor armazenado em *bytes*. O comando `cudaMemcpy` realiza a transferência entre as memórias da CPU e GPU. O primeiro argumento é o destino dos dados, o segundo, a origem dos dados, o terceiro, o número de *bytes* a serem transferidos e o quarto o sentido da transferência dos dados

(`cudaMemcpyHostToDevice` transfere os dados da CPU para a GPU e `cudaMemcpyDeviceToHost` transfere os dados da GPU para a CPU).

Algoritmo 5 Código em linguagem C do incremento dos elementos de um vetor.

```
void inc_cpu(int *a, int N)
{
    int idx;
    for (idx = 0; idx < N; idx++)
        a[idx] = a[idx] + 1;
}
void main()
{
    inc_cpu(a, N);
}
```

Algoritmo 6 Código em linguagem CUDA do incremento dos elementos de um vetor.

```
__global__ void inc_gpu(int *a_d, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
        a_d[idx] = a_d[idx] + 1;
}
void main()
{
    dim3 dimBlock(blocksize);
    dim3 dimGrid(ceil( N / (float)blocksize));
    inc_gpu<<<dimGrid, dimBlock>>>(a_d, N);
}
```

Em processos em que a mesma posição da memória global é acessada mais de uma vez pelas *threads* do mesmo bloco, torna-se interessante o uso da memória compartilhada. O Algoritmo 8 apresenta duas formas de utilização da memória compartilhada. No `kernel1` a memória compartilhada é alocada estaticamente possuindo um tamanho fixo. No `kernel2` a alocação da memória compartilhada ocorre de forma dinâmica ao declarar a variável como `extern`.

Um fator importante no uso da memória compartilhada é a sincronização (com o comando `__syncthreads()`) que se faz necessária em determinadas situações. Isto pode ser observado no `kernel1` em que valores do mesmo vetor são modificados e posteriormente utilizados, fazendo-se necessária a sincronização para que os valores sejam transferidos corretamente, como desejado.

Algoritmo 7 Código em linguagem CUDA do uso da memória global pela CPU.

```

void main()
{ ...
  float *a_h;
  __device__ float *a_d;
  cudaMalloc((void **)&a_d, N * sizeof(float));

  cudaMemcpy(a_d, a_h, numBytes, cudaMemcpyHostToDevice);

  inc_gpu<<<ceil(N/(float)blocksize), blocksize>>>(a_d, N);

  cudaMemcpy(a_h, a_d, numBytes, cudaMemcpyDeviceToHost);
... }

```

Algoritmo 8 Código em linguagem CUDA do uso da memória global pela GPU.

```

__global__ void kernel1(...)
{
...
  __shared__ float sData[256];

  sData[idx] = sData[idx] + threadIdx.x;
  __syncthreads();
  sData[idx] = sData[idx + 1];
...
}

__global__ void kernel2(...)
{
...
  extern __shared__ float sData[];
...
}

int main(void)
{
...
  kernel1<<< nBlocks, blockSize >>>(...);
  smBytes=blockSize*sizeof(float);
  kernel2<<< nBlocks, blockSize, smBytes >>>(...);
...
}

```

APÊNDICE B – ALGORITMOS DO MHS

O Algoritmo 9 mostra o fluxo principal de controle do algoritmo MHS implementada em linguagem C. Neste algoritmo os passos do MHS são bem definidos representados por um único método.

Algoritmo 9 Código em linguagem C do fluxo principal de controle da MHS.

```
void harmony_search()
{
    /* Passo 1: inicialização dos parâmetros */
    initialize_parameters();
    /* Passo 2: inicialização da memória harmônica */
    initialize_harmony_memory();
    int i = 0;
    do
    {
        /* Passo 3: Improviso de uma nova harmonia */
        improvise_new_harmony();
        /* Passo 4: Atualização da memória harmônica */
        update_HM(PS);
        i++;
        /* Passo 5: Verificação do critério de parada */
    } while(i <= MI);
    copyResultToHost();
    report_final();
}
```

Os algoritmos seguintes representam os passos do MHS com o trecho de código que realiza a chamada do *kernel*, seguido do código do respectivo *kernel* CUDA. O passo de inicialização da memória harmônica é realizado pelos Algoritmos 10 e 11. O passo de improviso é realizado pelos Algoritmos 12 e 11. E o Algoritmo 13 apresenta o processo de atualização da memória harmônica.

Algoritmo 10 Código em linguagem CUDA da primeira parte do processo de inicialização da memória harmônica.

```

RandomGPU<<< 32, 128 >>>(device_random, 2);
initialize_hm<<< N, (HMS + PS) >>> (limits_min, limits_max, device_random,
                                     HM, HM_fitness, random_index);
random_index += N * (MHS + PS);

...

__global__ void initialize_hm(float *limits_min, float *limits_max,
                             float *device_random, float *HM, float *HM_fitness, int random_index)
{
    float lmin = limits_min[blockIdx.x];
    float lmax = limits_max[blockIdx.x];
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    HM[idx] = device_random[(random_index + idx) % 4096] *
              (lmax - lmin) + lmin;

    if (blockIdx.x == 0)
    {
        HM_fitness[threadIdx.x] = 1e20;
        device_neval = 0;
    }
}

```

Algoritmo 11 Código em linguagem CUDA da segunda parte do processo de inicialização da memória harmônica.

```

for (i = 0; i < HMS; i++) {
    if (random_index > 4096 - N)
    {
        RandomGPU<<< 32, 128 >>>(device_random, 2);
        random_index = 0;
    }
    initialize_harmony_memory_device<<< 1, N >>>(limits_min, limits_max,
        HM, device_random, random_index, HMS);

    random_index += N;
    update_HM(1);
}

...

__global__ void initialize_harmony_memory_device( float *limits_min,
    float *limits_max, float *HM, float *device_random,
    int random_index, int HMS)
{
    int musician = threadIdx.x;
    float lmin = limits_min[N];
    float lmax = limits_max[N];
    /* blockDim.x = N */
    HM[HMS * blockDim.x + musician] = device_random[random_index + N] *
        (lmax - lmin) + lmin;
}

```

Algoritmo 12 Código em linguagem CUDA do processo de improviso de uma nova harmonia.

```

if (random_index > 4096 - 6 * PS) {
    RandomGPU<<<32, 128>>>(device_random, 2); // 2 * 4096 randoms
    random_index = 0;
}
improvise_new_harmony_device<<< N, PS, N >>>( limits_min, limits_max, HM,
    HM_fitness, device_random, random_index, HMCR, HMS, PS, PAR, FW, N);

random_index += 6 * PS;

...

__global__ void improvise_new_harmony_device( float *limits_min,
    float *limits_max, float *HM, float *HM_fitness, float *device_random,
    int random_index, float HMCR, int HMS, int PS, float PAR, float FW, int N )
{
    int i = blockIdx.x;

    int r_index = threadIdx.x * blockDim.x + random_index + i * 4;
    int aux = (threadIdx.x + HMS) * blockDim.x + blockIdx.x;
    float randHMCR = device_random[r_index];
    float lmin = limits_min[i];
    float lmax = limits_max[i];
    float note;
    int posHM = device_random[random_index] * (HMS + PS - 1)

    note = HM[posHM * blockDim.x + i];
    extern __shared__ float randPAR[];

    if (randHMCR <= HMCR)
    {
        if (randPAR[i] <= PAR)
        {
            float alpha=device_random[random_index+5]*2*FW-FW;
            if ((note+alpha >= lmin) && (note+alpha <= lmax))
                note = note + alpha;
        }
    } else {
        note = device_random[random_index + 1] *
            (lmax - lmin) + lmin;
    }

    __syncthreads();

    HM[aux] = note;
}

```

Algoritmo 13 Código em linguagem CUDA do processo de atualização da memória harmônica.

```

update_HM_device<<< N, HMS - 1 >>>(HM, HM_fitness, HM_neval,
                                   posicao_new, posicao);

...

__global__ void update_HM_device( float *HM, float *HM_fitness,
    int *HM_neval, int *posicao_new, int *posicao_insert ) {
    int musician = blockIdx.x;
    int index = threadIdx.x;
    int musicians = gridDim.x;
    int imm = index * N + musician;

    float aux = HM[imm];
    float aux_fitness = HM_fitness[index];
    float aux_neval = HM_neval[index];

    /* sincroniza para que todos as variáveis tenham
       sido copiadas antes do deslocamento */
    __syncthreads();

    if (index >= *posicao) {
        /* desloca harmonias, mantendo ordenadas */
        int index1 = index + 1;
        int imm1 = index1 * musicians + musician;
        HM[imm1] = aux;
        if (musician == 0) {
            HM_fitness[index1] = aux_fitness;
            HM_neval[index1] = aux_neval;
        }
    }
    if (index == *posicao) { /* insere o indivíduo */
        HM[imm] = HM[*posicao_insert * N + musician];
        if (musician == 0) {
            HM_fitness[*posicao_insert] = HM_fitness[*posicao_new];
            HM_neval[*posicao_insert] = HM_neval[*posicao_new];
        }
    }
}
}
}

```

APÊNDICE C - ANÁLISES ESTATÍSTICAS

As análises estatísticas apresentadas aqui foram realizadas utilizando o aplicativo Sisvar, desenvolvido na Universidade Federal de Lavras – UFLA.

C.1 IDENTIFICAÇÃO DE PARÂMETROS - PRIMEIRA PARTE

Aqui são apresentadas as análises estatísticas realizadas na primeira etapa de identificação de parâmetros apresentada na Seção 4.2.

Teste de Normalidade

Estatísticas descritivas básicas	
n:	9719
média aritmética amostral:	-3.70681
variância:	0.19798
desvio padrão:	0.44495
desvio padrão não viesado:	0.44496
coeficiente de variação(em %):	-12.00356
erro padrão da média:	0.00451
soma total:	-36026.45155
soma de quadrados não corrig.:	135467.04377
soma de quadrados corrigida:	1923.96204
amplitude total (A ou R):	3.84180
mínimo:	-6.09824
máximo:	-2.25644
amplitude estudentizada (W):	8.63426

Obs. O Desvio padrão não viesado é obtido por: $S_c = Q * S$, em que Q é dado por: $Q = [(n - 1)/2]^{0.5} * G[(n - 1)/2] / G[n/2] * S$; sendo G(a) a função gama do argumento a.

Nota: Os estimadores beta de assimetria e curtose têm como referências os valores 0 (para o coef. de assimetria) e 3 para o de curtose. Já os estimadores gama têm como referência o valor 0 (zero). Os momentos foram divididos por (n-1) e não por n.

Obs. O teste de Kolmogorov-Smirnov deve ser visto com reserva para pequenos tamanhos de amostras - uma vez que a média e a variância foram estimados dos dados.

Estatísticas da natureza da distribuição	
Coef. de Assimetria - Estimador beta:	-0.81200
Coef. de Assimetria - Estimador gama:	-0.81212
Coef. de curtose - Estimador beta:	5.50028
Coef. de curtose - Estimador gama:	2.50219
Momento de ordem 3 centrado na média:	-0.07153
Momento de ordem 4 centrado na média:	0.21557

Teste de normalidade	
Kolmogorov-Smirnov: $D = 0.07430$ $pr < D = 0.00000$	

A seguir é apresentado o resultado da análise de variância e teste de Scott-Knott para as estratégias de blocos construtivos maiores (BLOCK), a utilização de uma harmonia base (BASE), a utilização de um método de seleção (TOURN), a explosão ou dizimação (EXPLOSION)

Variável analisada: MEDIA

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	Pr > Fc
TOURN	2	1.816278	0.908139	4.681	0.0093
BLOCK	2	1.341514	0.670757	3.457	0.0316
BASE	1	10.511206	10.511206	54.177	0.0000
EXPLOSION	2	0.255432	0.127716	0.658	0.5178
erro	9386	1821.029269	0.194015		
Total corrigido	9719	1923.962168			

CV (%) = -11.88

Média geral: -3.7068076

Número de observações: 9720

Teste Scott-Knott (1974) para a FV TOURN

NMS: 0.05

Média harmonica do número de repetições (r): 3240

Erro padrão: 0.00773830214715405

Tratamentos	Médias	Resultados do teste
5	-3.726102	a1
3	-3.698207	a2
1	-3.696114	a2

Teste Scott-Knott (1974) para a FV BLOCK

NMS: 0.05

Média harmonica do número de repetições (r): 3240

Erro padrão: 0.00773830214715405

Tratamentos	Médias	Resultados do teste
1	-3.718464	a1
5	-3.711232	a1
3	-3.690726	a2

Teste Scott-Knott (1974) para a FV BASE

NMS: 0.05

Média harmonica do número de repetições (r): 4860

Erro padrão: 0.00631829724533696

Tratamentos	Médias	Resultados do teste
3	-3.739692	a1
0	-3.673923	a2

Teste Scott-Knott (1974) para a FV EXPLOSION

NMS: 0.05

Média harmonica do número de repetições (r): 3240

Erro padrão: 0.00773830214715405

Tratamentos	Médias	Resultados do teste
∞	-3.710644	a1
1000	-3.710217	a1
10000	-3.699562	a1

C.2 IDENTIFICAÇÃO DE PARÂMETROS - SEGUNDA PARTE

Aqui são apresentadas as análises estatísticas realizadas na segunda etapa de identificação de parâmetros apresentada na Seção 4.2.

Variável analisada: MEDIA

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	<i>Pr > Fc</i>
MBL	1	3.506138713E+0010	3.50613871E+0010	3572.188	0.0000
EXP	1	1.690062794E+0010	1.69006279E+0010	1721.900	0.0000
erro	717	7.037427809E+0009	9815101.546074		
Total corrigido	719	5.899944288E+0010			

CV (%) = 5.36

Média geral: 58420.1550417

Número de observações: 720

Teste Scott-Knott (1974) para a FV MBL

NMS: 0.05

Média harmonica do número de repetições (r): 360

Erro padrão: 304.320328846169

Tratamentos	Médias	Resultados do teste
5	51441.876528	a1
1	65398.433556	a2

Teste Scott-Knott (1974) para a FV EXP

NMS: 0.05

Média harmonica do número de repetições (r): 360

Erro padrão: 165.118657217515

Tratamentos	Médias	Resultados do teste
10000000	53575.251083	a1
1000	63265.059000	a2

C.3 ANÁLISE DAS ESTRATÉGIAS PROPOSTAS

Aqui são apresentadas as análises estatísticas das estratégias propostas apresentadas na Seção 4.6. Comparando o comportamento do uso individual de cada estratégia em diferentes situações.

A seguir são mostrados os resultados da análise de variância e teste de Scott-Knott para o uso do método de seleção torneio estocástico.

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	Pr > Fc
TS	2	0.175807	0.087904	0.476	0.6229
erro	87	16.067728	0.184687		

Total corrigido 89 16.243535

CV (%) = -11.82

Média geral: -3.6348089

Número de observações: 90

Teste Scott-Knott (1974) para a FV TS

NMS: 0.05

Média harmonica do número de repetições (r): 30

Erro padrão: 0.0784615666046517

Tratamentos	Médias	Resultados do teste
5	-3.691207	a1
3	-3.629946	a1
0	-3.583274	a1

A seguir são mostrados os resultados da análise de variância e teste de Scott-Knott para o uso de diferentes tamanhos máximos de blocos construtivos.

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	Pr > Fc
MBL	3	0.299102	0.099701	0.826	0.4822
erro	116	14.006474	0.120745		

Total corrigido 119 14.305576

CV (%) = -9.76

Média geral: -3.5587073

Número de observações: 120

Teste Scott-Knott (1974) para a FV MBL

NMS: 0.05

Média harmonica do número de repetições (r): 30

Erro padrão: 0.0634416963772727

Tratamentos	Médias	Resultados do teste
5	-3.595606	a1
1	-3.583274	a1
3	-3.472675	a1

A seguir são mostrados os resultados da análise de variância e teste de Scott-Knott para o uso ou não de uma harmonia base.

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	<i>Pr > Fc</i>
UBHR	1	0.075385	0.075385	0.699	0.4066
erro	58	6.256735	0.107875		

Total corrigido 59 6.332120

CV (%) = -9.08

Média geral: -3.6187200

Número de observações: 60

Teste Scott-Knott (1974) para a FV UBHR

NMS: 0.05

Média harmonica do número de repetições (r): 30

Erro padrão: 0.0599651974277724

Tratamentos	Médias	Resultados do teste
0.3	-3.654166	a1
0	-3.583274	a1

A seguir são mostrados os resultados da análise de variância e teste de Scott-Knott para o uso de diferentes intervalos para explosão.

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	<i>Pr > Fc</i>
EXP	2	0.310185	0.155093	1.057	0.3518
erro	87	12.760906	0.146677		

Total corrigido 89 13.071092

CV (%) = -10.71

Média geral: -3.5757484

Número de observações: 90

Teste Scott-Knott (1974) para a FV EXP

NMS: 0.05

Média harmonica do número de repetições (r): 30

Erro padrão: 0.0699230734694021

Tratamentos	Médias	Resultados do teste
1.000	-3.643591	a1
∞	-3.583274	a1
10.000	-3.500381	a1

A seguir são mostrados os resultados da análise de variância e teste de Scott-Knott para o uso de diferentes estratégias de autoadaptação.

TABELA DE ANÁLISE DE VARIÂNCIA					
FV	GL	SQ	QM	Fc	<i>Pr > Fc</i>
Auto	2	0.339251	0.169626	0.538	0.5858
erro	87	27.422836	0.315205		

Total corrigido 89 27.762088

CV (%) = -15.51

Média geral: -3.6192144

Número de observações: 90

Teste Scott-Knott (1974) para a FV Auto

NMS: 0.05

Média harmonica do número de repetições (r): 30

Erro padrão: 0.10250284771077

Tratamentos	Médias	Resultados do teste
A0	-3.705635	a1
SA	-3.583274	a1
A1	-3.568735	a1

APÊNDICE D – RESULTADOS DOS EXPERIMENTOS FATORIAIS

D.1 DADOS EXPERIMENTAIS PARA AJUSTE DE PARÂMETROS

D.1.1 Problema de dobramento de Proteínas, 21 aminoácidos

Na Tabela 9 são apresentados os experimentos fatoriais executados em CPU (1 a 6) e em GPU (7 a 324) para o problema de dobramento de proteínas AB-2D, para a instância de 21 aminoácidos. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.2, na seção 4.3 e na seção 4.5.

Tabela 9 – Resultados de 324 experimentos realizados para o problema de otimização de dobramento de proteínas AB-2D para a sequencia de Fibonacci de 21 aminoácidos

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCRC	PAR	Tempo(s)	Média	Melhor
1	32	SA	1	0	1	∞	80	20	16,70	-3,73	-6,02
2	32	SA	1	0	1	∞	80	30	16,69	-3,61	-4,58
3	32	SA	1	0	1	∞	90	20	16,70	-3,72	-5,18
4	32	SA	1	0	1	∞	90	30	16,69	-3,56	-5,01
5	32	SA	1	0	1	∞	99	20	16,70	-3,69	-4,81
6	32	SA	1	0	1	∞	99	30	16,69	-3,72	-5,76
7	32	SA	1	0	1	1000	80	20	16,71	-3,67	-5,04
8	32	SA	1	0	1	1000	80	30	16,69	-3,75	-4,67
9	32	SA	1	0	1	1000	90	20	16,70	-3,60	-5,09
10	32	SA	1	0	1	1000	90	30	16,69	-3,57	-4,50
11	32	SA	1	0	1	1000	99	20	16,70	-3,73	-5,16
12	32	SA	1	0	1	1000	99	30	16,69	-3,69	-4,68
13	32	SA	1	0	1	10000	80	20	16,70	-3,62	-4,29
14	32	SA	1	0	1	10000	80	30	16,69	-3,61	-4,59
15	32	SA	1	0	1	10000	90	20	16,71	-3,72	-5,69
16	32	SA	1	0	1	10000	90	30	16,69	-3,64	-5,09
17	32	SA	1	0	1	10000	99	20	16,70	-3,65	-4,73
18	32	SA	1	0	1	10000	99	30	16,69	-3,57	-4,92
19	32	SA	1	0	3	∞	80	20	17,50	-3,80	-5,85

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
20	32	SA	1	0	3	∞	80	30	17,48	-3,80	-5,97
21	32	SA	1	0	3	∞	90	20	17,50	-3,81	-5,32
22	32	SA	1	0	3	∞	90	30	17,48	-3,75	-5,04
23	32	SA	1	0	3	∞	99	20	17,50	-3,73	-4,95
24	32	SA	1	0	3	∞	99	30	17,48	-3,71	-5,99
25	32	SA	1	0	3	1000	80	20	17,50	-3,80	-5,08
26	32	SA	1	0	3	1000	80	30	17,49	-3,70	-4,60
27	32	SA	1	0	3	1000	90	20	17,50	-3,73	-5,02
28	32	SA	1	0	3	1000	90	30	17,49	-3,70	-5,12
29	32	SA	1	0	3	1000	99	20	17,50	-3,78	-5,26
30	32	SA	1	0	3	1000	99	30	17,48	-3,73	-4,70
31	32	SA	1	0	3	10000	80	20	17,50	-3,68	-5,25
32	32	SA	1	0	3	10000	80	30	17,49	-3,73	-4,59
33	32	SA	1	0	3	10000	90	20	17,50	-3,63	-5,35
34	32	SA	1	0	3	10000	90	30	17,48	-3,74	-4,94
35	32	SA	1	0	3	10000	99	20	17,50	-3,67	-4,85
36	32	SA	1	0	3	10000	99	30	17,49	-3,79	-4,65
37	32	SA	1	0	5	∞	80	20	18,24	-3,77	-4,67
38	32	SA	1	0	5	∞	80	30	18,23	-3,74	-4,47
39	32	SA	1	0	5	∞	90	20	18,24	-3,75	-5,67
40	32	SA	1	0	5	∞	90	30	18,23	-3,71	-4,73

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
41	32	SA	1	0	5	∞	99	20	18,24	-3,85	-5,99
42	32	SA	1	0	5	∞	99	30	18,23	-3,78	-5,71
43	32	SA	1	0	5	1000	80	20	18,24	-3,78	-4,94
44	32	SA	1	0	5	1000	80	30	18,22	-3,70	-4,67
45	32	SA	1	0	5	1000	90	20	18,24	-3,81	-5,13
46	32	SA	1	0	5	1000	90	30	18,22	-3,68	-4,74
47	32	SA	1	0	5	1000	99	20	18,24	-3,69	-4,72
48	32	SA	1	0	5	1000	99	30	18,22	-3,76	-5,01
49	32	SA	1	0	5	10000	80	20	18,25	-3,78	-4,65
50	32	SA	1	0	5	10000	80	30	18,23	-3,78	-5,93
51	32	SA	1	0	5	10000	90	20	18,25	-3,92	-6,00
52	32	SA	1	0	5	10000	90	30	18,23	-3,85	-5,94
53	32	SA	1	0	5	10000	99	20	18,25	-3,62	-5,92
54	32	SA	1	0	5	10000	99	30	18,23	-3,71	-4,58
55	32	SA	1	0,3	1	∞	80	20	16,69	-3,62	-5,00
56	32	SA	1	0,3	1	∞	80	30	16,69	-3,61	-4,95
57	32	SA	1	0,3	1	∞	90	20	16,69	-3,70	-4,62
58	32	SA	1	0,3	1	∞	90	30	16,69	-3,59	-4,42
59	32	SA	1	0,3	1	∞	99	20	16,69	-3,60	-4,39
60	32	SA	1	0,3	1	∞	99	30	16,69	-3,60	-4,39
61	32	SA	1	0,3	1	1000	80	20	16,70	-3,79	-5,69

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
62	32	SA	1	0,3	1	1000	80	30	16,70	-3,59	-4,18
63	32	SA	1	0,3	1	1000	90	20	16,69	-3,70	-5,42
64	32	SA	1	0,3	1	1000	90	30	16,70	-3,61	-4,74
65	32	SA	1	0,3	1	1000	99	20	16,70	-3,65	-4,73
66	32	SA	1	0,3	1	1000	99	30	16,70	-3,63	-4,52
67	32	SA	1	0,3	1	10000	80	20	16,69	-3,68	-4,89
68	32	SA	1	0,3	1	10000	80	30	16,69	-3,57	-4,41
69	32	SA	1	0,3	1	10000	90	20	16,70	-3,70	-4,48
70	32	SA	1	0,3	1	10000	90	30	16,70	-3,53	-4,61
71	32	SA	1	0,3	1	10000	99	20	16,70	-3,75	-4,80
72	32	SA	1	0,3	1	10000	99	30	16,69	-3,64	-4,82
73	32	SA	1	0,3	3	∞	80	20	17,50	-3,74	-4,92
74	32	SA	1	0,3	3	∞	80	30	17,48	-3,73	-4,82
75	32	SA	1	0,3	3	∞	90	20	17,50	-3,64	-4,68
76	32	SA	1	0,3	3	∞	90	30	17,48	-3,67	-4,33
77	32	SA	1	0,3	3	∞	99	20	17,50	-3,71	-5,13
78	32	SA	1	0,3	3	∞	99	30	17,48	-3,70	-4,49
79	32	SA	1	0,3	3	1000	80	20	17,49	-3,66	-4,87
80	32	SA	1	0,3	3	1000	80	30	17,48	-3,67	-4,79
81	32	SA	1	0,3	3	1000	90	20	17,50	-3,76	-4,87
82	32	SA	1	0,3	3	1000	90	30	17,48	-3,64	-5,13

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
83	32	SA	1	0,3	3	1000	99	20	17,50	-3,67	-4,67
84	32	SA	1	0,3	3	1000	99	30	17,48	-3,64	-4,34
85	32	SA	1	0,3	3	10000	80	20	17,49	-3,80	-6,06
86	32	SA	1	0,3	3	10000	80	30	17,48	-3,70	-4,97
87	32	SA	1	0,3	3	10000	90	20	17,49	-3,81	-5,18
88	32	SA	1	0,3	3	10000	90	30	17,48	-3,76	-5,11
89	32	SA	1	0,3	3	10000	99	20	17,49	-3,72	-5,04
90	32	SA	1	0,3	3	10000	99	30	17,48	-3,74	-5,14
91	32	SA	1	0,3	5	∞	80	20	18,24	-3,78	-4,90
92	32	SA	1	0,3	5	∞	80	30	18,22	-3,71	-5,62
93	32	SA	1	0,3	5	∞	90	20	18,23	-3,75	-4,95
94	32	SA	1	0,3	5	∞	90	30	18,22	-3,71	-4,99
95	32	SA	1	0,3	5	∞	99	20	18,25	-3,77	-6,10
96	32	SA	1	0,3	5	∞	99	30	18,22	-3,68	-5,28
97	32	SA	1	0,3	5	1000	80	20	18,25	-3,78	-5,14
98	32	SA	1	0,3	5	1000	80	30	18,23	-3,70	-4,84
99	32	SA	1	0,3	5	1000	90	20	18,25	-3,68	-5,88
100	32	SA	1	0,3	5	1000	90	30	18,23	-3,67	-5,20
101	32	SA	1	0,3	5	1000	99	20	18,24	-3,76	-4,73
102	32	SA	1	0,3	5	1000	99	30	18,23	-3,70	-5,79
103	32	SA	1	0,3	5	10000	80	20	18,24	-3,69	-5,35

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
104	32	SA	1	0,3	5	10000	80	30	18,22	-3,76	-5,19
105	32	SA	1	0,3	5	10000	90	20	18,26	-3,77	-5,09
106	32	SA	1	0,3	5	10000	90	30	18,24	-3,71	-5,16
107	32	SA	1	0,3	5	10000	99	20	18,26	-3,76	-5,11
108	32	SA	1	0,3	5	10000	99	30	18,24	-3,63	-4,76
109	32	SA	3	0	1	∞	80	20	16,72	-3,72	-6,06
110	32	SA	3	0	1	∞	80	30	16,72	-3,60	-4,64
111	32	SA	3	0	1	∞	90	20	16,73	-3,72	-5,02
112	32	SA	3	0	1	∞	90	30	16,72	-3,65	-5,11
113	32	SA	3	0	1	∞	99	20	16,73	-3,76	-5,20
114	32	SA	3	0	1	∞	99	30	16,74	-3,62	-4,32
115	32	SA	3	0	1	1000	80	20	16,76	-3,74	-5,07
116	32	SA	3	0	1	1000	80	30	16,76	-3,62	-4,85
117	32	SA	3	0	1	1000	90	20	16,76	-3,72	-5,88
118	32	SA	3	0	1	1000	90	30	16,75	-3,59	-4,47
119	32	SA	3	0	1	1000	99	20	16,76	-3,73	-5,63
120	32	SA	3	0	1	1000	99	30	16,75	-3,60	-4,68
121	32	SA	3	0	1	10000	80	20	16,72	-3,66	-4,94
122	32	SA	3	0	1	10000	80	30	16,71	-3,62	-4,72
123	32	SA	3	0	1	10000	90	20	16,72	-3,71	-5,25
124	32	SA	3	0	1	10000	90	30	16,72	-3,63	-5,12

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
125	32	SA	3	0	1	10000	99	20	16,72	-3,68	-5,07
126	32	SA	3	0	1	10000	99	30	16,72	-3,54	-4,38
127	32	SA	3	0	3	∞	80	20	17,52	-3,69	-5,08
128	32	SA	3	0	3	∞	80	30	17,51	-3,71	-4,86
129	32	SA	3	0	3	∞	90	20	17,52	-3,77	-5,71
130	32	SA	3	0	3	∞	90	30	17,51	-3,77	-5,24
131	32	SA	3	0	3	∞	99	20	17,52	-3,69	-5,14
132	32	SA	3	0	3	∞	99	30	17,50	-3,64	-4,93
133	32	SA	3	0	3	1000	80	20	17,52	-3,79	-6,06
134	32	SA	3	0	3	1000	80	30	17,51	-3,75	-5,77
135	32	SA	3	0	3	1000	90	20	17,52	-3,79	-5,85
136	32	SA	3	0	3	1000	90	30	17,51	-3,64	-4,99
137	32	SA	3	0	3	1000	99	20	17,52	-3,83	-5,86
138	32	SA	3	0	3	1000	99	30	17,51	-3,71	-4,79
139	32	SA	3	0	3	10000	80	20	17,52	-3,57	-5,34
140	32	SA	3	0	3	10000	80	30	17,51	-3,75	-5,18
141	32	SA	3	0	3	10000	90	20	17,52	-3,80	-4,98
142	32	SA	3	0	3	10000	90	30	17,51	-3,72	-5,95
143	32	SA	3	0	3	10000	99	20	17,52	-3,68	-4,81
144	32	SA	3	0	3	10000	99	30	17,51	-3,69	-5,16
145	32	SA	3	0	5	∞	80	20	18,26	-3,73	-5,73

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
146	32	SA	3	0	5	∞	80	30	18,24	-3,79	-5,03
147	32	SA	3	0	5	∞	90	20	18,26	-3,69	-5,31
148	32	SA	3	0	5	∞	90	30	18,25	-3,80	-5,70
149	32	SA	3	0	5	∞	99	20	18,26	-3,73	-5,22
150	32	SA	3	0	5	∞	99	30	18,25	-3,73	-5,18
151	32	SA	3	0	5	1000	80	20	18,27	-3,77	-5,86
152	32	SA	3	0	5	1000	80	30	18,24	-3,80	-5,06
153	32	SA	3	0	5	1000	90	20	18,26	-3,91	-5,92
154	32	SA	3	0	5	1000	90	30	18,24	-3,73	-4,93
155	32	SA	3	0	5	1000	99	20	18,26	-3,70	-5,11
156	32	SA	3	0	5	1000	99	30	18,24	-3,65	-6,05
157	32	SA	3	0	5	10000	80	20	18,27	-3,84	-5,91
158	32	SA	3	0	5	10000	80	30	18,25	-3,69	-5,31
159	32	SA	3	0	5	10000	90	20	18,27	-3,69	-4,86
160	32	SA	3	0	5	10000	90	30	18,25	-3,78	-5,93
161	32	SA	3	0	5	10000	99	20	18,26	-3,74	-5,18
162	32	SA	3	0	5	10000	99	30	18,24	-3,71	-4,92
163	32	SA	3	0,3	1	∞	80	20	16,71	-3,68	-5,25
164	32	SA	3	0,3	1	∞	80	30	16,71	-3,59	-4,63
165	32	SA	3	0,3	1	∞	90	20	16,71	-3,74	-5,82
166	32	SA	3	0,3	1	∞	90	30	16,71	-3,64	-5,09

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
167	32	SA	3	0,3	1	∞	99	20	16,71	-3,70	-5,45
168	32	SA	3	0,3	1	∞	99	30	16,71	-3,72	-4,63
169	32	SA	3	0,3	1	1000	80	20	16,72	-3,71	-4,47
170	32	SA	3	0,3	1	1000	80	30	16,72	-3,68	-4,68
171	32	SA	3	0,3	1	1000	90	20	16,72	-3,79	-5,31
172	32	SA	3	0,3	1	1000	90	30	16,71	-3,59	-4,88
173	32	SA	3	0,3	1	1000	99	20	16,72	-3,77	-4,70
174	32	SA	3	0,3	1	1000	99	30	16,71	-3,65	-4,57
175	32	SA	3	0,3	1	10000	80	20	16,72	-3,69	-5,45
176	32	SA	3	0,3	1	10000	80	30	16,62	-3,65	-5,02
177	32	SA	3	0,3	1	10000	90	20	16,62	-3,73	-5,89
178	32	SA	3	0,3	1	10000	90	30	16,62	-3,64	-4,72
179	32	SA	3	0,3	1	10000	99	20	16,62	-3,65	-4,40
180	32	SA	3	0,3	1	10000	99	30	16,63	-3,66	-4,52
181	32	SA	3	0,3	3	∞	80	20	17,96	-3,70	-4,96
182	32	SA	3	0,3	3	∞	80	30	17,94	-3,69	-5,25
183	32	SA	3	0,3	3	∞	90	20	17,95	-3,69	-5,20
184	32	SA	3	0,3	3	∞	90	30	17,95	-3,66	-4,73
185	32	SA	3	0,3	3	∞	99	20	17,96	-3,58	-4,57
186	32	SA	3	0,3	3	∞	99	30	17,95	-3,61	-5,19
187	32	SA	3	0,3	3	1000	80	20	17,95	-3,64	-4,45

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
188	32	SA	3	0,3	3	1000	80	30	17,94	-3,83	-6,03
189	32	SA	3	0,3	3	1000	90	20	17,95	-3,74	-5,67
190	32	SA	3	0,3	3	1000	90	30	17,94	-3,78	-4,85
191	32	SA	3	0,3	3	1000	99	20	17,95	-3,64	-4,89
192	32	SA	3	0,3	3	1000	99	30	17,94	-3,73	-5,72
193	32	SA	3	0,3	3	10000	80	20	17,96	-3,77	-5,79
194	32	SA	3	0,3	3	10000	80	30	17,94	-3,79	-5,18
195	32	SA	3	0,3	3	10000	90	20	17,96	-3,76	-5,46
196	32	SA	3	0,3	3	10000	90	30	17,95	-3,75	-5,63
197	32	SA	3	0,3	3	10000	99	20	17,96	-3,68	-4,98
198	32	SA	3	0,3	3	10000	99	30	17,95	-3,62	-4,75
199	32	SA	3	0,3	5	∞	80	20	19,26	-3,84	-5,91
200	32	SA	3	0,3	5	∞	80	30	19,23	-3,70	-4,55
201	32	SA	3	0,3	5	∞	90	20	19,26	-3,71	-4,96
202	32	SA	3	0,3	5	∞	90	30	19,23	-3,76	-5,11
203	32	SA	3	0,3	5	∞	99	20	19,25	-3,79	-5,10
204	32	SA	3	0,3	5	∞	99	30	19,24	-3,68	-4,47
205	32	SA	3	0,3	5	1000	80	20	19,25	-3,71	-4,79
206	32	SA	3	0,3	5	1000	80	30	19,24	-3,73	-4,55
207	32	SA	3	0,3	5	1000	90	20	19,25	-3,79	-5,54
208	32	SA	3	0,3	5	1000	90	30	19,24	-3,77	-5,81

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
209	32	SA	3	0,3	5	1000	99	20	19,25	-3,80	-5,55
210	32	SA	3	0,3	5	1000	99	30	19,24	-3,64	-4,98
211	32	SA	3	0,3	5	10000	80	20	19,25	-3,93	-5,81
212	32	SA	3	0,3	5	10000	80	30	19,24	-3,68	-5,73
213	32	SA	3	0,3	5	10000	90	20	19,25	-3,84	-5,20
214	32	SA	3	0,3	5	10000	90	30	19,23	-3,68	-4,60
215	32	SA	3	0,3	5	10000	99	20	19,25	-3,79	-6,03
216	32	SA	3	0,3	5	10000	99	30	19,23	-3,68	-4,81
217	32	SA	5	0	1	∞	80	20	16,64	-3,75	-5,59
218	32	SA	5	0	1	∞	80	30	16,62	-3,58	-4,41
219	32	SA	5	0	1	∞	90	20	16,64	-3,72	-5,39
220	32	SA	5	0	1	∞	90	30	16,63	-3,55	-4,39
221	32	SA	5	0	1	∞	99	20	16,63	-3,70	-5,63
222	32	SA	5	0	1	∞	99	30	16,63	-3,60	-4,45
223	32	SA	5	0	1	1000	80	20	16,63	-3,69	-4,45
224	32	SA	5	0	1	1000	80	30	16,62	-3,55	-4,55
225	32	SA	5	0	1	1000	90	20	16,64	-3,69	-4,82
226	32	SA	5	0	1	1000	90	30	16,63	-3,61	-5,12
227	32	SA	5	0	1	1000	99	20	16,64	-3,74	-4,69
228	32	SA	5	0	1	1000	99	30	16,63	-3,66	-4,43
229	32	SA	5	0	1	10000	80	20	16,64	-3,65	-4,70

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
230	32	SA	5	0	1	10000	80	30	16,62	-3,58	-4,70
231	32	SA	5	0	1	10000	90	20	16,63	-3,65	-5,10
232	32	SA	5	0	1	10000	90	30	16,62	-3,53	-4,58
233	32	SA	5	0	1	10000	99	20	16,64	-3,73	-4,61
234	32	SA	5	0	1	10000	99	30	16,62	-3,53	-4,45
235	32	SA	5	0	3	∞	80	20	17,96	-3,82	-5,04
236	32	SA	5	0	3	∞	80	30	17,95	-3,71	-5,84
237	32	SA	5	0	3	∞	90	20	17,96	-3,74	-6,10
238	32	SA	5	0	3	∞	90	30	17,95	-3,61	-4,80
239	32	SA	5	0	3	∞	99	20	17,96	-3,69	-4,57
240	32	SA	5	0	3	∞	99	30	17,95	-3,70	-4,72
241	32	SA	5	0	3	1000	80	20	17,96	-3,84	-5,08
242	32	SA	5	0	3	1000	80	30	17,95	-3,70	-4,73
243	32	SA	5	0	3	1000	90	20	17,96	-3,84	-5,00
244	32	SA	5	0	3	1000	90	30	17,95	-3,74	-4,71
245	32	SA	5	0	3	1000	99	20	17,96	-3,74	-4,84
246	32	SA	5	0	3	1000	99	30	17,95	-3,63	-4,55
247	32	SA	5	0	3	10000	80	20	17,96	-3,73	-5,66
248	32	SA	5	0	3	10000	80	30	17,95	-3,64	-5,13
249	32	SA	5	0	3	10000	90	20	17,96	-3,69	-5,07
250	32	SA	5	0	3	10000	90	30	17,94	-3,73	-5,18

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
251	32	SA	5	0	3	10000	99	20	17,96	-3,69	-4,81
252	32	SA	5	0	3	10000	99	30	17,95	-3,67	-5,11
253	32	SA	5	0	5	∞	80	20	19,25	-3,88	-5,90
254	32	SA	5	0	5	∞	80	30	19,24	-3,73	-4,79
255	32	SA	5	0	5	∞	90	20	19,25	-3,76	-5,11
256	32	SA	5	0	5	∞	90	30	19,24	-3,79	-5,01
257	32	SA	5	0	5	∞	99	20	19,25	-3,82	-5,75
258	32	SA	5	0	5	∞	99	30	19,23	-3,70	-5,45
259	32	SA	5	0	5	1000	80	20	19,25	-3,85	-5,52
260	32	SA	5	0	5	1000	80	30	19,24	-3,91	-5,81
261	32	SA	5	0	5	1000	90	20	19,25	-3,84	-5,97
262	32	SA	5	0	5	1000	90	30	19,23	-3,80	-5,08
263	32	SA	5	0	5	1000	99	20	19,25	-3,74	-5,29
264	32	SA	5	0	5	1000	99	30	19,24	-3,76	-5,27
265	32	SA	5	0	5	10000	80	20	19,25	-3,80	-4,81
266	32	SA	5	0	5	10000	80	30	19,23	-3,71	-5,71
267	32	SA	5	0	5	10000	90	20	19,25	-3,83	-5,57
268	32	SA	5	0	5	10000	90	30	19,24	-3,74	-4,68
269	32	SA	5	0	5	10000	99	20	19,25	-3,78	-5,98
270	32	SA	5	0	5	10000	99	30	19,24	-3,77	-4,84
271	32	SA	5	0,3	1	∞	80	20	16,63	-3,69	-5,09

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
272	32	SA	5	0,3	1	∞	80	30	16,63	-3,68	-4,75
273	32	SA	5	0,3	1	∞	90	20	16,63	-3,75	-4,68
274	32	SA	5	0,3	1	∞	90	30	16,63	-3,57	-4,47
275	32	SA	5	0,3	1	∞	99	20	16,63	-3,69	-4,73
276	32	SA	5	0,3	1	∞	99	30	16,63	-3,63	-5,12
277	32	SA	5	0,3	1	1000	80	20	16,68	-3,61	-4,65
278	32	SA	5	0,3	1	1000	80	30	16,65	-3,66	-4,39
279	32	SA	5	0,3	1	1000	90	20	16,65	-3,66	-5,30
280	32	SA	5	0,3	1	1000	90	30	16,65	-3,60	-4,24
281	32	SA	5	0,3	1	1000	99	20	16,65	-3,73	-5,10
282	32	SA	5	0,3	1	1000	99	30	16,65	-3,55	-4,45
283	32	SA	5	0,3	1	10000	80	20	16,65	-3,73	-4,46
284	32	SA	5	0,3	1	10000	80	30	16,65	-3,63	-4,72
285	32	SA	5	0,3	1	10000	90	20	16,65	-3,71	-5,52
286	32	SA	5	0,3	1	10000	90	30	16,65	-3,64	-4,97
287	32	SA	5	0,3	1	10000	99	20	16,65	-3,76	-5,14
288	32	SA	5	0,3	1	10000	99	30	16,65	-3,60	-4,49
289	32	SA	5	0,3	3	∞	80	20	17,98	-3,75	-4,89
290	32	SA	5	0,3	3	∞	80	30	17,97	-3,72	-5,15
291	32	SA	5	0,3	3	∞	90	20	17,98	-3,81	-5,06
292	32	SA	5	0,3	3	∞	90	30	17,97	-3,65	-4,24

(continua)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
293	32	SA	5	0,3	3	∞	99	20	17,98	-3,77	-5,26
294	32	SA	5	0,3	3	∞	99	30	17,97	-3,71	-4,68
295	32	SA	5	0,3	3	1000	80	20	17,98	-3,74	-5,22
296	32	SA	5	0,3	3	1000	80	30	17,97	-3,76	-5,73
297	32	SA	5	0,3	3	1000	90	20	17,98	-3,74	-4,91
298	32	SA	5	0,3	3	1000	90	30	17,97	-3,72	-5,06
299	32	SA	5	0,3	3	1000	99	20	17,98	-3,70	-5,33
300	32	SA	5	0,3	3	1000	99	30	17,97	-3,77	-4,99
301	32	SA	5	0,3	3	10000	80	20	17,98	-3,90	-5,98
302	32	SA	5	0,3	3	10000	80	30	17,97	-3,61	-4,33
303	32	SA	5	0,3	3	10000	90	20	17,98	-3,77	-4,91
304	32	SA	5	0,3	3	10000	90	30	17,97	-3,79	-5,72
305	32	SA	5	0,3	3	10000	99	20	17,98	-3,71	-4,72
306	32	SA	5	0,3	3	10000	99	30	17,97	-3,60	-4,95
307	32	SA	5	0,3	5	∞	80	20	19,27	-3,72	-5,52
308	32	SA	5	0,3	5	∞	80	30	19,25	-3,77	-5,56
309	32	SA	5	0,3	5	∞	90	20	19,27	-3,73	-4,67
310	32	SA	5	0,3	5	∞	90	30	19,26	-3,74	-5,49
311	32	SA	5	0,3	5	∞	99	20	19,27	-3,74	-5,19
312	32	SA	5	0,3	5	∞	99	30	19,26	-3,75	-4,51
313	32	SA	5	0,3	5	1000	80	20	19,27	-3,73	-5,16

(conclusão)

#	PS	Auto	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
314	32	SA	5	0,3	5	1000	80	30	19,26	-3,81	-5,35
315	32	SA	5	0,3	5	1000	90	20	19,28	-3,77	-5,15
316	32	SA	5	0,3	5	1000	90	30	19,26	-3,70	-5,76
317	32	SA	5	0,3	5	1000	99	20	19,28	-3,75	-5,30
318	32	SA	5	0,3	5	1000	99	30	19,27	-3,71	-5,97
319	32	SA	5	0,3	5	10000	80	20	19,27	-3,78	-5,83
320	32	SA	5	0,3	5	10000	80	30	19,26	-3,76	-4,98
321	32	SA	5	0,3	5	10000	90	20	19,27	-3,77	-5,56
322	32	SA	5	0,3	5	10000	90	30	19,38	-3,70	-5,83
323	32	SA	5	0,3	5	10000	99	20	19,28	-3,78	-5,52
324	32	SA	5	0,3	5	10000	99	30	19,28	-3,71	-4,81

Fonte: Autoria própria.

D.1.2 Otimização estrutural de treliças, 200 barras

Na Tabela 10 são apresentados os experimentos fatoriais executados em CPU (325 a 330) e em GPU (331 a 354) para o problema de otimização estrutural de treliças, para a instância de 200 barras. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.2, na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 10 – Resultados de 24 experimentos realizados para o problema de otimização de estrutura de treliças para a instância de 200 barras

(continua)

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
325	1	1	0	1	∞	80	10	34,33	48740,6	47710,5
326	1	1	0	1	∞	80	20	25,43	49923,2	48977,3
327	1	1	0	1	∞	90	10	25,43	49465,4	48060,6
328	1	1	0	1	∞	90	20	25,43	49861,1	48419,1
329	1	1	0	1	∞	99	10	25,43	49569,5	48543,2
330	1	1	0	1	∞	99	20	25,43	49641,6	47952,7
331	32	1	0,3	5	1000	80	10	3,81	72931,8	67026,5
332	32	1	0,3	5	1000	80	20	4,79	72904,0	66881,0
333	32	1	0,3	5	1000	90	10	5,08	72515,1	65021,6
334	32	1	0,3	5	1000	90	20	5,08	73063,3	67446,4
335	32	1	0,3	5	1000	99	10	5,08	72475,4	66458,9
336	32	1	0,3	5	1000	99	20	5,08	72864,6	67674,5
337	32	1	0,3	5	∞	80	10	3,81	59802,3	55665,5
338	32	1	0,3	5	∞	80	20	4,87	57841,4	53766,4
339	32	1	0,3	5	∞	90	10	5,08	57688,5	53983,4
340	32	1	0,3	5	∞	90	20	5,08	57463,3	54007,9
341	32	1	0,3	5	∞	99	10	5,08	57250,4	53826,6
342	32	1	0,3	5	∞	99	20	5,08	57981,0	54593,1
343	32	5	0,3	5	1000	80	10	3,82	53097,8	51325,0
344	32	5	0,3	5	1000	80	20	5,01	54601,1	52261,7
345	32	5	0,3	5	1000	90	10	5,09	52902,9	50884,7
346	32	5	0,3	5	1000	90	20	5,09	54195,1	52354,4
347	32	5	0,3	5	1000	99	10	5,09	53306,8	51489,4
348	32	5	0,3	5	1000	99	20	5,09	54322,9	52729,4
349	32	5	0,3	5	∞	80	10	3,82	49148,4	47679,8

(conclusão)

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
350	32	5	0,3	5	∞	80	20	5,09	49424,4	48151,2
351	32	5	0,3	5	∞	90	10	5,09	48898,0	47016,0
352	32	5	0,3	5	∞	90	20	5,09	49175,8	48243,2
353	32	5	0,3	5	∞	99	10	5,09	48920,4	47063,7
354	32	5	0,3	5	$-\infty$	99	20	5,09	49309,1	48058,0

Fonte: Aatoria própria.

D.2 DADOS EXPERIMENTAIS DAS ANÁLISES DE RESULTADOS

D.2.1 Otimização estrutural de treliças, 10 barras

Na Tabela 11 são apresentados os experimentos fatoriais executados em CPU (355 a 360) e em GPU (361 a 366) para o problema de otimização estrutural de treliças, para a instância de 10 barras. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 11 – Resultados de 12 experimentos realizados para o problema de otimização estrutural de treliças para a instância de 10 barras

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
355	1	1	0	1	∞	80	10	0,88	5078,75	5065,59
356	1	1	0	1	∞	80	20	0,88	5073,85	5063,37
357	1	1	0	1	∞	90	10	0,88	5079,39	5064,01
358	1	1	0	1	∞	90	20	0,88	5074,42	5064,18
359	1	1	0	1	∞	99	10	0,88	5081,07	5067,48
360	1	1	0	1	∞	99	20	0,88	5074,70	5064,47
361	32	5	0,3	5	∞	80	10	0,42	5096,44	5081,40
362	32	5	0,3	5	∞	80	20	0,42	5088,72	5080,59
363	32	5	0,3	5	∞	90	10	0,42	5262,33	5088,51
364	32	5	0,3	5	∞	90	20	0,42	5085,94	5073,06
365	32	5	0,3	5	∞	99	10	0,42	5108,80	5091,21
366	32	5	0,3	5	∞	99	20	0,42	5086,93	5074,66

Fonte: Autoria própria.

D.2.2 Problema de dobramento de Proteínas, 13 aminoácidos

Na Tabela 12 são apresentados os experimentos fatoriais executados em CPU (367 a 372) e em GPU (373 a 378) para o problema de dobramento de proteínas AB-2D, para a instância de 13 aminoácidos. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 12 – Resultados de 12 experimentos realizados para o problema de dobramento de proteínas AB-2D para a sequência de Fibonacci de 13 aminoácidos

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
367	1	1	0	1	∞	80	10	683,94	2110,81	-1,03
368	1	1	0	1	∞	80	20	626,77	22232,07	-1,23
369	1	1	0	1	∞	90	10	625,41	1,21	-0,95
370	1	1	0	1	∞	90	20	624,21	402,10	-0,40
371	1	1	0	1	∞	99	10	622,10	2872,64	-1,08
372	1	1	0	1	∞	99	20	621,92	736,16	-1,46
373	32	5	0,3	5	∞	80	10	34,67	0,14	-0,93
374	32	5	0,3	5	∞	80	20	34,66	-0,64	-1,47
375	32	5	0,3	5	∞	90	10	34,67	0,06	-1,43
376	32	5	0,3	5	∞	90	20	34,67	-1,02	-2,42
377	32	5	0,3	5	∞	99	10	34,68	-0,08	-1,31
378	32	5	0,3	5	∞	99	20	34,67	-0,99	-2,24

Fonte: Autoria própria.

D.2.3 Problema de dobramento de Proteínas, 34 aminoácidos

Na Tabela 13 são apresentados os experimentos fatoriais executados em CPU (379 a 384) e em GPU (385 a 390) para o problema de dobramento de proteínas AB-2D, para a instância de 34 aminoácidos. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 13 – Resultados de 12 experimentos realizados para o problema de dobramento de proteínas AB-2D para a sequência de Fibonacci de 34 aminoácidos

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
379	1	1	0	1	∞	80	10	92,04	-2,33	-3,18
380	1	1	0	1	∞	80	20	92,64	-2,78	-3,22
381	1	1	0	1	∞	90	10	92,12	-2,73	-3,18
382	1	1	0	1	∞	90	20	92,99	-2,79	-3,15
383	1	1	0	1	∞	99	10	92,80	-2,63	-3,26
384	1	1	0	1	∞	99	20	92,66	-2,52	-3,15
385	32	5	0,3	5	∞	80	10	17,63	-2,02	-2,78
386	32	5	0,3	5	∞	80	20	17,64	-1,90	-2,76
387	32	5	0,3	5	∞	90	10	17,63	-2,10	-3,14
388	32	5	0,3	5	∞	90	20	17,64	-1,98	-2,73
389	32	5	0,3	5	∞	99	10	17,63	-2,10	-2,87
390	32	5	0,3	5	∞	99	20	17,64	-1,91	-2,50

Fonte: Autoria própria.

D.2.4 Problema matemático Griewank com 30 dimensões

Na Tabela 14 são apresentados os experimentos fatoriais executados em CPU (391 a 396) e em GPU (397 a 402) para o problema matemático Griewank com 30 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 14 – Resultados de 12 experimentos realizados para a otimização do problema matemático Griewank com 30 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
391	1	1	0	1	∞	80	10	11,97	0,016	0,0002
392	1	1	0	1	∞	80	20	12,54	0,042	0,0203
393	1	1	0	1	∞	90	10	11,94	0,013	0,0002
394	1	1	0	1	∞	90	20	12,54	0,040	0,0146
395	1	1	0	1	∞	99	10	11,89	0,011	0,0001
396	1	1	0	1	∞	99	20	12,51	0,038	0,0206
397	32	1	0,3	5	∞	80	10	2,43	0,052	0,0173
398	32	1	0,3	5	∞	80	20	2,52	0,672	0,5343
399	32	1	0,3	5	∞	90	10	2,42	0,049	0,0160
400	32	1	0,3	5	∞	90	20	2,51	0,716	0,4682
401	32	1	0,3	5	∞	99	10	2,42	0,049	0,0174
402	32	1	0,3	5	∞	99	20	2,51	0,696	0,4164

Fonte: Autoria própria.

D.2.5 Problema matemático Griewank com 50 dimensões

Na Tabela 15 são apresentados os experimentos fatoriais executados em CPU (403 a 408) e em GPU (409 a 414) para o problema matemático Griewank com 50 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 15 – Resultados de 12 experimentos realizados para a otimização do problema matemático Griewank com 50 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
403	1	1	0	1	∞	80	10	18,12	0,02	0,007
404	1	1	0	1	∞	80	20	18,93	0,95	0,841
405	1	1	0	1	∞	90	10	18,19	0,03	0,009
406	1	1	0	1	∞	90	20	18,95	0,96	0,783
407	1	1	0	1	∞	99	10	18,15	0,02	0,007
408	1	1	0	1	∞	99	20	18,96	0,95	0,756
409	32	1	0,3	5	∞	80	10	2,77	0,70	0,525
410	32	1	0,3	5	∞	80	20	2,88	1,09	1,069
411	32	1	0,3	5	∞	90	10	2,77	0,73	0,590
412	32	1	0,3	5	∞	90	20	2,87	1,10	1,063
413	32	1	0,3	5	∞	99	10	2,77	0,68	0,490
414	32	1	0,3	5	∞	99	20	2,87	1,09	1,069

Fonte: Autoria própria.

D.2.6 Problema matemático Rosenbrock com 30 dimensões

Na Tabela 16 são apresentados os experimentos fatoriais executados em CPU (415 a 420) e em GPU (421 a 426) para o problema matemático Rosenbrock com 30 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 16 – Resultados de 12 experimentos realizados para a otimização do problema matemático Rosenbrock com 30 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
415	1	1	0	1	∞	80	10	10,44	0,31	0,004
416	1	1	0	1	∞	80	20	10,33	0,42	0,023
417	1	1	0	1	∞	90	10	10,33	0,24	0,004
418	1	1	0	1	∞	90	20	10,57	0,42	0,042
419	1	1	0	1	∞	99	10	10,30	0,20	0,001
420	1	1	0	1	∞	99	20	10,59	0,62	0,002
421	32	5	0,3	5	∞	80	10	3,15	2,68	0,001
422	32	5	0,3	5	∞	80	20	3,20	14,15	0,032
423	32	5	0,3	5	∞	90	10	3,15	2,48	0,017
424	32	5	0,3	5	∞	90	20	3,19	27,98	0,014
425	32	5	0,3	5	∞	99	10	3,14	3,74	0,010
426	32	5	0,3	5	∞	99	20	3,19	20,64	0,007

Fonte: Autoria própria.

D.2.7 Problema matemático Rosenbrock com 50 dimensões

Na Tabela 17 são apresentados os experimentos fatoriais executados em CPU (427 a 432) e em GPU (433 a 439) para o problema matemático Rosenbrock com 50 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 17 – Resultados de 12 experimentos realizados para a otimização do problema matemático Rosenbrock com 50 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
427	1	1	0	1	∞	80	10	16,06	7,56	0,00
428	1	1	0	1	∞	80	20	16,34	17,16	0,02
429	1	1	0	1	∞	90	10	16,03	5,35	0,01
430	1	1	0	1	∞	90	20	16,39	41,08	0,08
431	1	1	0	1	∞	99	10	16,12	17,87	0,00
432	1	1	0	1	∞	99	20	16,30	26,86	0,02
433	32	5	0,3	5	∞	80	10	3,61	50,45	0,02
434	32	5	0,3	5	∞	80	20	3,68	59,94	0,19
435	32	5	0,3	5	∞	90	10	3,62	39,60	0,07
436	32	5	0,3	5	∞	90	20	3,68	63,07	1,70
437	32	5	0,3	5	∞	99	10	3,65	37,61	0,10
438	32	5	0,3	5	∞	99	20	3,66	66,59	2,27

Fonte: Autoria própria.

D.2.8 Problema matemático Schaffer com 30 dimensões

Na Tabela 18 são apresentados os experimentos fatoriais executados em CPU (439 a 444) e em GPU (445 a 450) para o problema matemático Schaffer com 30 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 18 – Resultados de 12 experimentos realizados para a otimização do problema matemático Schaffer com 30 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
439	1	1	0	1	∞	80	10	6,90	1,02	0,50
440	1	1	0	1	∞	80	20	7,25	1,37	1,00
441	1	1	0	1	∞	90	10	6,86	1,13	0,53
442	1	1	0	1	∞	90	20	7,22	1,54	0,86
443	1	1	0	1	∞	99	10	6,85	0,99	0,40
444	1	1	0	1	∞	99	20	7,21	1,39	1,02
445	32	5	0,3	5	∞	80	10	1,96	1,80	0,72
446	32	5	0,3	5	∞	80	20	2,03	3,74	2,56
447	32	5	0,3	5	∞	90	10	1,95	1,48	0,67
448	32	5	0,3	5	∞	90	20	2,26	1,72	0,78
449	32	5	0,3	5	∞	99	10	1,98	1,84	1,04
450	32	5	0,3	5	∞	99	20	2,04	3,59	2,67

Fonte: Aatoria própria.

D.2.9 Problema matemático Schaffer com 50 dimensões

Na Tabela 19 são apresentados os experimentos fatoriais executados em CPU (451 a 456) e em GPU (457 a 462) para o problema matemático Rosenbrock com 50 dimensões. Estes dados são usados como base para as análises realizadas e informações apresentadas na seção 4.3, na seção 4.4 e na seção 4.5.

Tabela 19 – Resultados de 12 experimentos realizados para a otimização do problema matemático Schaffer com 50 dimensões

#	PS	MBL	UBHR	TS	EXP	HMCR	PAR	Tempo(s)	Média	Melhor
451	1	1	0	1	∞	80	10	10,85	3,94	2,97
452	1	1	0	1	∞	80	20	11,32	6,59	5,12
453	1	1	0	1	∞	90	10	10,91	3,98	2,87
454	1	1	0	1	∞	90	20	11,40	6,71	4,55
455	1	1	0	1	∞	99	10	10,92	4,15	2,66
456	1	1	0	1	∞	99	20	11,36	6,75	4,90
457	32	5	0,3	5	∞	80	10	2,31	6,76	5,15
458	32	5	0,3	5	∞	80	20	2,39	11,78	10,12
459	32	5	0,3	5	∞	90	10	2,31	6,86	5,48
460	32	5	0,3	5	∞	90	20	2,38	11,85	8,86
461	32	5	0,3	5	∞	99	10	2,31	7,02	4,80
462	32	5	0,3	5	∞	99	20	2,39	11,81	9,66

Fonte: Aatoria própria.

