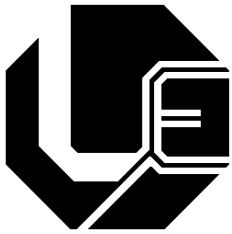
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



Estudo da Influência dos Parâmetros de Algoritmos Paralelos da Computação Evolutiva no seu Desempenho em Plataformas *Multicore*

Mônica Sakuray Pais

UBERLÂNDIA
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Mônica Sakuray Pais

Estudo da Influência dos Parâmetros de Algoritmos Paralelos da Computação Evolutiva no seu Desempenho em Plataformas *Multicore*

Texto da tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para a obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação, Inteligência Artificial

Orientador: Prof. Dr. Keiji Yamanaka
Co-orientador: Prof. Dr. Edmilson R. Pinto

UBERLÂNDIA

2014

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU , MG, Brasil

P149e
2014

Pais, Mônica Sakuray, 1965-
Estudo da influência dos parâmetros de algoritmos paralelos da computação evolutiva no seu desempenho em plataformas Multicore / Mônica Sakuray Pais. - 2014.
214 p. : il.
Orientador: Keiji Yamanaka.
Coorientador: Edmilson Rodrigues Pinto.

Tese (doutorado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Algoritmos paralelos - Teses. 3. Planejamento experimental - Teses. I. Yamanaka, Keiji. II. Pinto, Edmilson Rodrigues. III. Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDU: 621.3

Mônica Sakuray Pais

Estudo da Influência dos Parâmetros de Algoritmos Paralelos da Computação Evolutiva no seu Desempenho em Plataformas *Multicore*

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como parte dos requisito para a obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação, Inteligência Artificial

Uberlândia, 14 de março de 2014

Banca Examinadora

Orientador - Prof. Dr. Keiji Yamanaka - FEELT-UFU

Co-orientador - Prof. Dr. Edmilson Rodrigues Pinto - FAMAT-UFU

Prof. Dr. Omar Andres Carmona Cortes - IFEMA

Prof. Dr. Felipe Campelo Franca Pinto - UFMG

Prof. Dra. Rita Maria da Silva Julia - FACOM-UFU

A José Carlos, meu esposo,
pelo amor, compreensão e
companheirismo.
Aos nossos filhos João e Carlos,
pelo amor e por todos
os momentos compartilhados.

Agradecimentos

Ao meu marido José Carlos e aos nossos filhos João e Carlos, pelo amor, suporte, carinho e compreensão durante esta complicada fase de dedicação à minha pesquisa.

Aos meus pais, pelo amor, apoio e por sempre acreditarem em mim.

Ao meu orientador, professor Dr. Keiji Yamanaka, pela confiança em mim depositada, pela presteza em auxiliar e pela grande oportunidade de trabalharmos juntos.

Ao meu co-orientador, professor Dr. Edmilson R. Pinto, pelas orientações e valiosas contribuições à este trabalho.

Aos colegas e amigos do Laboratório de Inteligência Computacional, Igor, Gerson, Josy, Hugo, Ricardo e Leonardo, pelo apoio e companheirismo.

Ao pessoal do Laboratório de Fontes Alternativas de Energia, em especial ao professor Dr. José Roberto Camacho por compartilhar o espaço e as conversas.

Aos companheiros de viagens e de discussões sobre esta pesquisa, Júlio César Ferreira e Fernando Barbosa Matos.

Ao Programa de Pós-Graduação da Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, em especial à Cinara Fagundes, sempre prestativa, eficiente e amiga.

Ao IFGOIANO, ao diretor do campus Urutaí Prof. Dr. Gilson Dourado e colegas professores, pelo apoio e confiança em mim depositada.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Fundação de Amparo à Pesquisa do Estado de Goiás (FAPEG) pelo financiamento parcial dessa pesquisa.

Resumo

A computação paralela é um modo poderoso de reduzir o tempo de processamento e de melhorar a qualidade das soluções dos algoritmos evolutivos (AE). No princípio, os AE paralelos (AEP) eram executados em máquinas paralelas caras e pouco disponíveis. Desde que os processadores *multicore* tornaram-se largamente disponíveis, sua capacidade de processamento paralelo é um grande incentivo para que os AE, programas exigentes de poder computacional, sejam paralelizados e explorem ao máximo a capacidade de processamento dos *multicore*. A implementação paralela traz mais fatores que podem influenciar a performance dos AEP e adiciona mais complexidade na avaliação desses algoritmos. A estatística pode ajudar nessa tarefa e garantir conclusões corretas e significativas, com o mínimo de testes, se for aplicado o planejamento de experimentos adequado. Neste trabalho é apresentada uma metodologia de experimentação com AEP. Essa metodologia garante a correta estimação do *speedup* e aplica o planejamento fatorial na análise dos fatores que influenciam o desempenho. Como estudo de caso, um algoritmo genético, denominado AGP-I, foi paralelizado segundo o modelo de ilhas. O AGP-I foi executado em plataformas com diferentes processadores *multicore* na resolução de duas funções de teste. A metodologia de experimentação com AEP foi aplicada para se determinar a influência dos fatores relacionados à migração no desempenho do AGP-I.

Palavras-chave

Algoritmos Evolutivos Paralelos, Processadores *Multicore*, Planejamento de Experimentos e Planejamento Fatorial.

Abstract

Parallel computing is a powerful way to reduce the computation time and to improve the quality of solutions of evolutionary algorithms (EAs). At first, parallel evolutionary algorithms (PEAs) ran on very expensive and not easily available parallel machines. As multicore processors become ubiquitous, the improved performance available to parallel programs is a great motivation to computationally demanding EAs to turn into parallel programs and exploit the power of multicores. The parallel implementation brings more factors to influence performance, and consequently adds more complexity on PEAs evaluations. Statistics can help in this task and guarantee the significance and correct conclusions with minimum tests, provided that the correct design of experiments is applied. This work presents a methodology that guarantees the correct estimation of speedups and applies a factorial design on the analysis of PEAs performance. As a case study, the influence of migration related parameters on the performance of a parallel evolutionary algorithm solving two benchmark problems executed on a multicore processor is evaluated.

Keywords

Parallel Evolutionary Algorithms, Multicore Processors, Design of Experiments, Factorial Design.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AE Algoritmos Evolutivos

AEP Algoritmo Evolutivo Paralelo

AG Algoritmos Genéticos

AIC Akaike information criterion

ALU *Arithimetic Logic Unit*

CC-NUMA *Coherent Cache - Non Uniform Memory Access*

CMP *Chip Multiprocessor*

COW *Cluster of Workstations*

CPU *Central Processing Unit*

DMA *Direct Memory Access*

DSM *Distributed Shared Memory*

GPU Unidade de Processamento Gráfico

HT *Hyper-Threading*

ILP *Instruction Level Parallelism*

MIMD *Multiple Instruction Stream, Multiple Data Stream*

MISD *Multiple Instruction Stream, Single Data Stream*

MPI *Message Passing Interface*

MPP *Massive Parallel Processor*

NC-NUMA *Noncoherent Cache - Non Uniform Memory Access*

NoC *Network On Chip*

NUMA *Non-Uniform Memory Access*

OpenMP *Open Multiprocessing*

PRNG *Pseudo Random Numbers Generators*

SIMD *Single Instruction Stream, Multiple Data Stream*

SISD *Single Instruction Stream, Single Data Stream*

SMP *Symmetric Shared Memory*

SMT *Simultaneous Multithreading*

TLP *Thread Level Parallelism*

UMA *Uniform Memory Access*

VLIW *Very Long Instruction Word*

Capítulo 1

Introdução

Existem inúmeros problemas reais cujas soluções dependem de estratégias de otimização de um determinado recurso finito. Exemplos de problemas desse tipo são: o escalonamento e roteamento de veículos de uma distribuidora de bebidas que maximize o número de entregas, o projeto de uma rede de irrigação que minimize o gasto de água e energia, projetos de redes de circuitos integrados, entre outros. Esses problemas são chamados de problemas de otimização combinatória e possuem um número grande de soluções possíveis, mesmo quando são impostas restrições para encontrar a solução desejada. A resolução desses problemas através da verificação de todas as soluções possíveis mostra-se inviável na prática. Para se resolver problemas com essas características é necessário ter um grande poder de processamento computacional e técnicas que possibilitem a obtenção de soluções de forma eficaz e eficiente. Dentre essas técnicas estão as metaheurísticas.

Metaheurísticas são algoritmos de aproximação da solução de problemas de otimização combinatória que combinam heurísticas em uma forma mais genérica. Algumas das metaheurísticas mais conhecidas são: os algoritmos de colônica de formigas, os algoritmos evolutivos (AE), os algoritmos de busca local iterativa, os algoritmos de recozimento simulado e a busca tabu.

Os AE são caracterizados por uma população de soluções candidatas sujeitas a operadores de variação com hereditariedade, sobre as quais é aplicada uma pressão seletiva. São inspirados no processo evolutivo biológico, onde indivíduos mais aptos têm maiores chances de sobrevivência e de se reproduzir. Dentre os diferentes AE propostos têm-se a programação evolutiva, as estratégias evolutivas e os algoritmos

genéticos (AG).

Os AE são resolvedores efetivos de problemas de otimização para os quais não existe método eficiente conhecido (Chiarandini et al., 2007). É comum que os AE requeiram grande esforço computacional e já existem tentativas de melhorar o seu desempenho através da sua paralelização (Cantú-Paz, 2000; Tomassini, 2005). De fato, os AE paralelos (AEP) podem atingir *speedups* superlineares (Alba, 2002).

Computadores paralelos têm sido sinônimo de supercomputadores, equipamentos grandes e caros, disponíveis apenas em alguns poucos centros de pesquisa. Desde que a indústria de processadores iniciou a fabricação dos processadores *multicore*, arquiteturas *multicore* encontram-se em vários dispositivos computadorizados, de sistemas embarcados aos computadores pessoais, tornando a computação paralela acessível a todos (Kim e Bond, 2009). Essa acessibilidade é um grande incentivo para a conversão de AE em AEP.

Porém, a complexidade dos AEP é uma desvantagem. Essa complexidade vem basicamente das dificuldades inerentes ao paralelismo e à avaliação dos AEP. São inúmeros os fatores que podem influenciar o desempenho, como os parâmetros dos AEP que são constituídos pelos parâmetros de um AE canônico acrescidos dos parâmetros introduzidos pelo paralelismo. Alguns desses parâmetros são qualitativos e outros quantitativos. Os parâmetros quantitativos podem assumir infinitos valores, tornando grande o número de combinações possíveis e, consequentemente, dificultando a tarefa de configuração dos AEP.

As avaliações de desempenho de algoritmos estocásticos, como os AEP, são estimadas ao se executar o algoritmo repetidas vezes para se obter um tempo de execução médio. Além disso, avaliações de algoritmos paralelos adotam uma medida de desempenho largamente utilizada, o *speedup*. No contexto da computação paralela, o *speedup* é calculado pela divisão do tempo de execução do programa sequencial pelo tempo de execução do programa paralelo. Para os AEP, o *speedup* é a razão entre a média dos tempos das execuções sequenciais e a média dos tempos das execuções paralelas, ou seja, a razão entre duas variáveis aleatórias independentes com média positiva. Essa razão nem sempre possui uma média bem definida (Díaz-Francés e Rubio, 2013; Qiao et al., 2006).

O desempenho dos AEP varia devido a diversos fatores, onde fator é uma variável independente, ou seja, uma variável manipulada em experimento cuja presença ou

nível determina a mudança no resultado. Esses fatores podem ser classificados como fatores dos AE, fatores do paralelismo, fatores da plataforma computacional e fatores do problema a ser solucionado.

Se existe o interesse em estudar como esses fatores podem influenciar o desempenho do AEP, é necessário executar experimentos onde mudanças intencionais nesses fatores são realizadas, de modo que seja possível observar e identificar as razões para as variações que ocorrem nos resultados. Um método comum de se avaliar desempenhos é conhecido como "um-fator-de-cada-vez". Esse método avalia a influência de cada fator, variando um de cada vez e mantendo os outros constantes. Não é possível identificar as interações entre os fatores através desse método. Outro método muito usado é o teste exaustivo de todos os fatores e todas as combinações de todos os níveis. É um método completo, mas caro. Esses dois métodos são usados quando o número de fatores e o número de níveis dos fatores é pequeno. A realização de experimentos e a análise dos resultados acarretam em algumas questões que devem ser respondidas, como:

- Qual metodologia de experimentação com algoritmos deve ser adotada? Existe uma metodologia padrão que pode ser adotada?
- Como provar que um dado algoritmo A é melhor que outro algoritmo B?
- Comparar apenas resultados médios dos AEP é suficiente para uma conclusão?

A estatística pode ajudar a responder as questões anteriores de forma racional e econômica. Uma das partes mais importantes da estatística é planejar os experimentos que produzirão os dados. Com experimentos bem planejados é possível extrair conclusões válidas, e a análise dos resultados passa a ser direta (Neto et al., 2010).

O planejamento experimental estatístico–em inglês, *Design of Experiments (DOE)*– representa um conjunto de ensaios estabelecidos com critérios científicos e estatísticos, com o objetivo de determinar a influência de diversas variáveis nos resultados de um sistema ou processo. A utilização dessa técnica permite: (1) determinar quais variáveis são mais influentes no resultado; (2) atribuir valores às variáveis influentes de modo a otimizar a resposta média; (3) atribuir valores às variáveis influentes de modo a minimizar a variabilidade dos resultados. A observação de um sistema em funcionamento e a experimentação podem fornecer informações importantes sobre como esse sistema trabalha. O processo de primeiro planejar um experimento para então coletar e analisar

esses dados através de métodos estatísticos que possibilitam extrair conclusões válidas e objetivas, evidencia a existência de dois aspectos em qualquer estudo experimental: o planejamento do experimento e a análise estatística dos dados. Esses dois aspectos estão intimamente relacionados, pois o método de análise depende diretamente do planejamento empregado (Montgomery, 2009).

O planejamento experimental estatístico do tipo fatorial é uma estratégia na qual fatores são variados simultaneamente, ao invés de um de cada vez. O planejamento fatorial 2^k é recomendado quando existem muitos fatores a serem investigados, e busca-se descobrir quais fatores e quais interações entre fatores são as mais influentes na resposta do experimento.

1.1 Objetivos

O objetivo geral deste trabalho é estudar a influência dos parâmetros de um AEP no seu desempenho em plataformas *multicore*. Para atingir esse objetivo, foi necessário elaborar uma metodologia baseada no planejamento experimental estatístico para estudar a influência dos parâmetros de um AEP executado em plataforma com processador *multicore*.

1.2 Contribuições

As principais contribuições deste trabalho são descritas a seguir:

- Revisão de trabalhos publicados sobre a metodologia de experimentação com algoritmos.

- Correta estimação do *speedup* como medida de desempenho dos AEP.

- Método para avaliação da influência de fatores no desempenho de AEP, onde o planejamento fatorial 2^k é aplicado para estimar o efeito dos fatores e suas interação.

- Estudo de caso da aplicação do método proposto no estudo da influência dos parâmetros da migração no desempenho de um AE, paralelizado segundo o modelo de ilhas, para a resolução das funções Rosenbrock e Rastrigin.

1.3 Metodologia

O desenvolvimento deste trabalho está dividido nas seguintes etapas:

- Revisão bibliográfica sobre os seguintes tópicos: processadores *multicore*, programação paralela, algoritmos genéticos e algoritmos genéticos paralelos, estudo de parâmetros de metaheurísticas e planejamento experimental.
- Implementação de um AEP para ser utilizado no estudo de caso.
- Definição da metodologia de estudo da influência de parâmetros dos AEP no seu desempenho em plataforma *multicore*.
- Estudo de caso: aplicação da metodologia proposta no estudo dos parâmetros de um AEP na resolução de funções de teste em diferentes plataformas *multicore*.
- Análise dos dados obtidos.
- Conclusão da metodologia.

1.4 Organização do Texto

Este texto contém, além deste capítulo de introdução com o contexto, a motivação e os objetivos deste trabalho, capítulos com o referencial teórico que introduzem as teorias e definições que fundamentam este trabalho e são importantes para a compreensão do mesmo, capítulos com os trabalhos relacionados e apresentação dos resultados obtidos.

O referencial teórico deste trabalho é abordado nos Capítulos 2, 3 e 4. O Capítulo 2 apresenta os algoritmos genéticos e discorre sobre os algoritmos genéticos paralelos, seus modelos e sua fundamentação teórica. O Capítulo 3 mostra como os processadores *multicore* tornaram-se a estratégia de evolução adotada pela indústria de microprocessadores, e as implicações que essa nova tecnologia traz, como a paralelização das aplicações. Nesse capítulo também são abordados aspectos da arquitetura dos sistemas paralelo e da programação paralela. O Capítulo 4 introduz técnicas de planejamento experimental, que são ferramentas importantes para obter do sistema em estudo o máximo de informação útil através de experimentos executados de forma racional e econômica.

No Capitulo 5, encontra-se a revisão da literatura das diversas abordagens para o estudo da influência dos parâmetros de metaheurísticas no seu desempenho. Como o

foco deste trabalho é a aplicação de técnicas estatísticas de planejamento e análise de experimentos, maior destaque é dado aos trabalhos que utilizam essas técnicas.

O estudo do desempenho de um algoritmo possui duas abordagens: a análise teórica e a análise experimental. No Capítulo 6, discorre-se sobre a análise experimental de algoritmos com ênfase na avaliação dos AE. A partir dessa revisão é apresentado o método para avaliação da influência de fatores no desempenho dos AEP, através da aplicação do planejamento estatístico fatorial 2^k.

O Capítulo 7 apresenta os resultados obtidos com aplicação do método de avaliação nos estudos de caso da avaliação da influência dos parâmetros de migração no desempenho de um AEP para as funções Rosenbrock e Rastrigin.

As considerações finais e as sugestões para os trabalhos futuros, complementares às contribuições deste estudo, são apresentadas no Capítulo 8.

1.5 Publicações

Este trabalho iniciou-se no ano de 2009, quando foram cursadas as disciplinas: Reconhecimento de Padrões, Redes Neurais Artificiais e Tópicos Especiais em Engenharia de Computação IV - Algoritmos Genéticos. A investigação da aplicação dos AE na solução de problemas gerou as seguintes publicações:

- Ferreira, J. C.; Pais, M. S.; Yamanaka, K.; Carrijo, G. A. . *Previsão de Vazão da Bacia do Ribeirão João Leite Utilizando Redes Neurais com Treinamento Levenberg-Marquardt.* In: IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional (IX CBRN), 2009, Ouro Preto. Anais do IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional (IX CBRN) (CD-Rom) ISSN 2177-1200. Rio de Janeiro, 2009.

- Pais, M. S.; Ferreira, J. C.; Teixeira, M. B.; Yamanaka, K.; Carrijo, G. A. . *Cost Optimization of a Localized Irrigation System Using Genetic Algorithms.* In: 11th International Conference on Intelligent Data Engineering and Automated Learning, 2010, IDEAL 2010. Heidelberg : Lecture Notes in Computer Science, Springer Berlin, ISSN 0302-9743 ; 6283, ISBN 9783642153808; 9783642153815, 2010. v. 6283. p. 29-36.

- Ferreira, J. C.; Pais, M. S.; Yamanaka, K.; Carrijo, G. A.; Teixeira, M. B.; Silva, R. T.; Rabelo, C. G.. *Previsão de Vazão da Bacia do Ribeirão João Leite*

Utilizando Redes Neurais Artificiais. Revista Irriga (UNESP Botucatu), ISSN 1413-7895, v. 16, p. 339-350, 2011.

As investigações no ano de 2010 foram voltadas à programação paralela em plataformas *multicore*, para então focar nos AE e AEP.

Em 2011, foi implementada a versão paralela de um AG, segundo o modelo de ilhas, e os primeiros testes com planejamento experimental foram realizados. As investigações preliminares foram publicadas em:

- Pais, M. S. ; Peretta, I. S. ; Lima, G. F. M. ; Tavares, J. A. ; Rocha, H. X. ; Yamanaka, K. . *Análise de Fatores Determinantes no Desempenho dos Algoritmos Genéticos Paralelos em Processadores Multinúcleos.* In: X Congresso Brasileiro de Inteligência Computacional, 2011, Fortaleza.

No ano de 2012, os dados coletados com o planejamento experimental foram analisados utilizando o software estatístico R (R Core Team, 2012), cujo código fonte está disponível sob a licença GNU GPL, e as versões binárias pré-compiladas são fornecidas para *Windows*, *Macintosh*, e muitos sistemas operacionais *Unix-like*.

Em 2013, parte dos resultados obtidos foram descritos em artigo publicado no periódico Journal of the Brazilian Computer Society (JBCS):

- Pais, M. S. ; Peretta, I. S. ; Yamanaka, K.; Pinto, E. R. . *Factorial Design Analysis Applied to the Performance of Parallel Evolutionary Algorithms.* Journal of the Brazilian Computer Society 2014, 20:6, ISSN: 0104-6500 (print version) e ISSN: 1678-4804 (electronic version),DOI: 10.1186/10.1186/1678-4804-20-6.

E foi submetido artigo no periódico IEEE Latin America Transactions:

- Pais, M. S. ; Yamanaka, K.; Pinto, E. R. . *Rigorous Experimental Performance Analysis of Parallel Evolutionary Algorithms on Multicore Platforms*, IEEE Latin America Transactions, ISSN: 1548-0992.

Capítulo 2

Algoritmos Genéticos

Este Capítulo apresenta os conceitos e definições relacionados aos Algoritmos Genéticos e aos Algoritmos Genéticos Paralelos.

2.1 Introdução

Algoritmos Genéticos (AG) foram introduzidos por John Holland em 1975 e fazem parte dos Algoritmos Evolutivos (AE). Os AE constituem uma família de métodos computacionais inspirados na evolução natural das espécies aplicados a tarefas de busca e otimização. As tarefas de busca e otimização possuem como componentes básicos: o espaço de busca onde são consideradas todas as possíveis soluções de um determinado problema, e uma função objetivo que possibilita avaliar os membros do espaço de busca.

Os AG empregam uma terminologia originada da teoria da evolução natural e da genética. Um indivíduo da população é representado por um único cromossomo, o qual contém a codificação (genótipo) de uma possível solução do problema (fenótipo). Os cromossomos são um conjunto de atributos do indivíduo, onde cada atributo é chamado de gene. Cada gene pode assumir um conjunto de valores possíveis que são chamados de alelos.

Na fase inicial é gerada uma população formada por um conjunto aleatório de indivíduos que representam possíveis soluções do problema. Em seguida, durante o processo evolutivo, essa população é avaliada: para cada indivíduo é dado um valor refletindo sua habilidade de adaptação a determinado ambiente. Esse valor é utilizado pelo processo de seleção dos indivíduos que serão mantidos. Os indivíduos mantidos podem sofrer modificações através de mutações e do processo de reprodução. No processo de

reprodução ocorre o cruzamento (em inglês, *crossover*), no qual a recombinação genética produz descendentes para a próxima geração. Os processos de seleção, mutação e reprodução são repetidos até que uma solução satisfatória seja encontrada.

Quando um AG é aplicado a um problema de otimização, cada solução do problema deve ser codificada ou representada na forma de uma estrutura finita (vetor, matriz, etc.). Em seguida, devem ser definidos os operadores genéticos de seleção, cruzamento, mutação e estratégias de elitismo. Esses operadores devem ser escolhidos de acordo com as características intrínsecas do problema. Antes de aplicar um algoritmo genético para a solução de problema de otimização, vários parâmetros devem ser especificados, tais como, tamanho da população, probabilidade de cruzamento, probabilidade de mutação, dentre outros.

Os AG são métodos flexíveis e têm a capacidade de produzir soluções de boa qualidade para problemas complexos e de grande porte. Têm sido aplicados com sucesso em uma variedade de problemas em otimização combinatória NP-Completos e NP-Difíceis (Goldberg, 1989; Michalewicz, 1996).

Quando utilizados no contexto de otimização, os AG apresentam as seguintes vantagens em relação às técnicas clássicas de programação matemática (Goldberg, 1989):

- não necessitam de informações adicionais, como derivadas sobre a função objetivo, e, dessa forma, a região viável pode ser um conjunto convexo ou até mesmo disjunto, e a função objetivo pode possuir simultaneamente variáveis reais, lógicas e inteiras, ser não-convexa e não-diferenciável;

- empregam técnicas de transição probabilísticas sobre um conjunto de soluções, diversificando a busca;

- podem diminuir o risco de apresentar como solução um ponto ótimo local e, portanto, são mais adequados para trabalhar com problemas multimodais;

- são de fácil implementação, possibilitam uma grande flexibilidade em relação ao tratamento da função objetivo e apresentam um bom desempenho para uma grande escala de problemas do mundo real.

Além disso, os AG podem realizar a busca pela solução mantendo um equilíbrio entre dois objetivos aparentemente conflitantes: o aproveitamento das melhores soluções

e a exploração do espaço de busca (*exploitation* vs. *exploration*). AG procuram manter o equilíbrio entre a exploração de pontos inteiramente novos do espaço de busca, realizando uma busca em largura (*exploration*), isto é, manter a diversidade da população e utilizar a informação oriunda de pontos anteriormente visitados para encontrar os melhores, realizando uma busca em profundidade (*exploitation*). O cruzamento e mutação são dois mecanismos que levam à exploração de pontos inteiramente novos do espaço de busca. A seleção, por sua vez, dirige a busca em direção aos melhores pontos existentes no espaço de busca (ou população) (Lacerda e Carvalho, 2002).

Quando a aptidão é praticamente a mesma para toda a população, isto é, a pressão de seleção é muito baixa, os AG assumem um comportamento aleatório, pois não há seleção. Quando a pressão de seleção é muito alta, os AG assumem o comportamento dos métodos de Subida de Encosta (*Hill Climbing*).

Segundo Zuben (2000), uma característica importante dos algoritmos da computação evolutiva, incluindo os AG, está na possibilidade de resolver problemas pela simples descrição matemática do que se quer ver presente na solução, não havendo necessidade de se indicar explicitamente os passos até o resultado, que certamente seriam específicos para cada caso.

Entre as desvantagens para a utilização dos AG, têm-se:

- elevado custo computacional, uma vez que os AG necessitam de um grande número de avaliações da função objetivo para sua minimização (ou maximização);
- dificuldade para achar o ótimo global exato;
- numerosas opções de configurações de seus parâmetros, que podem complicar a resolução do problema sob estudo.

2.2 Algoritmo Genético Canônico

O Algoritmo 2.2.1 mostra um algoritmo genético simples ou canônico. Os critérios de parada mais comumente usados são: atingir número de gerações previamente definido, encontrar uma solução suficientemente boa ou a população não mais evoluir.

Nas próximas seções as várias etapas envolvidas na implementação de um algoritmo genético são apresentadas.

Algoritmo 2.2.1 Algoritmo Genético típico. Fonte: (Lacerda e Carvalho, 2002)

Seja $S(t)$ a população de cromossomos na geração t.

$t \leftarrow 0$
Inicializar $S(t)$
Avaliar $S(t)$
enquanto não satisfaz critério de parada **faça**
  $t \leftarrow t + 1$
  Selecionar $S(t)$ a partir de $S(t-1)$
  Aplicar cruzamento sobre $S(t)$
  Aplicar mutação sobre $S(t)$
  Avaliar $S(t)$
fim enquanto

2.2.1 Representação e Codificação do Indivíduo

A maneira mais simples de se representar um indivíduo é a representação binária de tamanho fixo, onde um indivíduo é uma cadeia de bits. A representação binária foi utilizada nos trabalhos pioneiros de Holland; é fácil de utilizar e manipular, como também, é simples de analisar teoricamente. Entretanto, há casos em que a representação binária não é a mais natural, nem a mais apropriada e, nesses casos, são indicados outros tipos de representação. Por exemplo, se um problema tem parâmetros contínuos e o usuário quer trabalhar com boa precisão numérica, cromossomos com representação binária corresponderiam a longas cadeias de bits, que além de ocupar mais espaço na memória, podem fazer o algoritmo convergir vagarosamente.

Por outro lado, a representação real em ponto flutuante gera cromossomos menores e é compreendida mais naturalmente, além de possibilitar a criação de novos operadores (Lacerda e Carvalho, 2002; Michalewicz, 1996).

2.2.2 Geração da População Inicial

A população inicial é gerada de modo aleatório. Utilizar algum tipo de heurística na geração da população inicial, como, acrescentar alguns indivíduos já conhecidos e bem avaliados ou definir uma distância mínima entre cromossomos, pode ser vantajoso em alguns casos. De modo geral, deve-se garantir a diversidade dos indivíduos, isto é, garantir que os indivíduos estejam uniformemente distribuídos por todo o espaço de busca.

O tamanho da população indica a quantidade de indivíduos na população. Em geral, esse tamanho permanece constante durante a evolução. Quanto maior a população,

maior a diversidade de soluções e maior o custo computacional. Esse custo computacional está relacionado com o número de avaliações de aptidão dos indivíduos. Portanto, o tamanho da população influencia diretamente o desempenho dos AG.

2.2.3 Avaliação da População

A avaliação de um determinado indivíduo se dá utilizando a função de aptidão, que fornece uma medida da qualidade da solução potencial desse indivíduo. A escolha da função de aptidão é a etapa crítica do processo para a maioria das aplicações. A função de aptidão e a codificação do indivíduo possuem relação direta com o domínio do problema.

2.2.4 Método de Seleção

Os AG devem ser capazes de identificar os indivíduos mais aptos, para que esses permaneçam na população durante o processo de evolução, e os menos aptos sejam excluídos do processo. Existem vários métodos de seleção propostos e implementados na prática, como: a seleção proporcional, a seleção por *ranking*, a seleção por torneio, dentre outros (Eiben e Smith, 2003).

- **Seleção Proporcional ou Método da Roleta.**
  Os indivíduos são preservados para a próxima geração de acordo com probabilidades proporcionais ao seu valor de função de aptidão. A implementação desse método é normalmente realizada através de um mecanismo de roleta no qual a roleta é dividida em N partes, N correspondendo ao número de indivíduos da população. O tamanho de cada uma das partes é proporcional ao valor da aptidão do indivíduo. A roleta é então girada N vezes e, a cada uma delas, o indivíduo indicado pelo ponteiro é selecionado e inserido na nova população.

- **Seleção por *Ranking*.**
  A seleção por *ranking* pode ser dividida em duas etapas. Na primeira, as soluções são ordenadas de acordo com seus valores da função de aptidão, em ordem crescente se o propósito for maximizar a função de aptidão ou em ordem decrescente caso o objetivo seja minimizá-la.

  Estando a lista ordenada, a cada indivíduo é atribuído um novo valor da função de aptidão equivalente à sua posição no ranking. Na segunda fase, um procedimento

similar à seleção proporcional é aplicado. Quanto melhor a posição do indivíduo no *ranking*, maior sua chance de ser selecionado.

- **Seleção por Torneio.**

  Este é um dos modelos mais simples para implementação computacional e apresenta bons resultados. A ideia é promover um torneio entre um grupo de indivíduos ($N > 1$) selecionados aleatoriamente a partir da população atual. Assim, o indivíduo com o maior valor de aptidão no grupo é selecionado, enquanto que os demais são descartados. A seleção por torneio não requer conhecimento global da população, não é necessária a ordenação dos indivíduos e a pressão seletiva pode ser controlada pelo valor de N, evitando a convergência prematura e combatendo a estagnação;

2.2.5 Elitismo

A estratégia de elitismo objetiva preservar e utilizar as melhores soluções encontradas na geração atual nas próximas gerações. Conserva-se uma quantidade n_e dos melhores indivíduos da população atual, copiando-os para a próxima geração sem nenhuma alteração. Os outros indivíduos da população são gerados normalmente, através do método de seleção e posterior aplicação dos operadores genéticos de cruzamento e mutação.

2.2.6 Operadores Genéticos

Os operadores genéticos são responsáveis pela transformação da população de geração em geração. Os AG canônicos têm dois operadores genéticos: cruzamento e mutação:

- **Operador Cruzamento.**

  O operador de cruzamento, também conhecido como *crossover*, realiza a troca de material genético entre dois indivíduos, combinando informações de maneira que exista uma probabilidade razoável dos novos indivíduos produzidos serem melhores que seus pais.

- **Operador Mutação.**

  A mutação é efetuada alterando-se o valor de um determinado gene de um indivíduo sorteado com uma determinada probabilidade, denominada probabilidade

de mutação, ou seja, vários indivíduos da nova população podem ter um de seus genes alterados aleatoriamente.

A operação de mutação é usada para garantir a diversidade da população e é aplicada em geral com uma probabilidade baixa. Uma probabilidade muito alta implica em uma busca aleatória.

2.2.7 Parâmetros dos Algoritmos Genéticos

Os parâmetros genéticos são grandezas que determinam o desempenho dos AG, adaptando-os às características particulares de determinada classe de problemas (Michalewicz, 1996). Entre eles os mais utilizados são: tamanho da população, o número de geração, a probabilidade de cruzamento e a probabilidade de mutação.

O número de parâmetros em um AE não é fixo, depende das escolha específicas de um dado projeto. No caso da escolha do método de seleção ser o método por torneio, é necessário informar também qual o tamanho do torneio. Se a escolha for pelo método da roleta, não há a necessidade de se adicionar outro parâmetro. Esse exemplo mostra que existe uma hierarquia entre os parâmetros (Eiben e Smith, 2011).

Principais parâmetros dos AG são:

- **Tamanho da População.**

  O tamanho da população indica o número de cromossomos em cada população e esse número normalmente se mantém constante durante a evolução.

- **Taxa ou Probabilidade de Cruzamento.**

  A probabilidade de cruzamento é uma grandeza percentual do número de indivíduos que experimentam o cruzamento em relação ao número total de indivíduos de uma população.

- **Taxa ou Probabilidade de Mutação.**

  A mutação fornece novas informações dentro da população, prevenindo que essa se torne saturada com indivíduos similares. A diversidade populacional aumenta com a mutação e faz com que haja uma maior varredura do espaço de busca. A taxa de mutação indica a probabilidade ou taxa em que haverá a mutação de cromossomos nas populações ao longo da evolução.

Além desses, é preciso definir quais são os métodos de cruzamento, mutação e seleção, o critério de parada, e parâmetros intrínsecos de cada método.

2.3 Algoritmos Genéticos Paralelos

Na década de 1980 os Algoritmos Genéticos Paralelos (AGP) aparecem em várias pesquisas com o objetivo de melhorar o desempenho através do uso dos supercomputadores e da computação distribuída, tecnologias recentes na época. A disponibilidade desses equipamentos era restrita devido ao seu alto custo econômico. No final da década de 1990, com o surgimento dos *clusters* construídos com computadores pessoais, a computação de alto desempenho e os AGP tornaram-se mais acessíveis a centros de pesquisa com orçamentos mais modestos. Ainda assim, o mais comum é encontrar implementações sequenciais de AG, abordagem que foi mantida com a ajuda da indústria de microprocessadores que garantiu o aumento do desempenho das aplicações ao lançar periodicamente processadores mais potentes que aumentavam o desempenho de qualquer aplicação. Atualmente, com os processadores *multicores*, o aumento de desempenho favorece os programas paralelos e são um incentivo à utilização do AGP.

A ideia básica, por trás da maioria dos programas paralelos, é dividir um problema grande em tarefas menores e executar essas tarefas simultaneamente, usando múltiplos processadores. Essa abordagem do tipo *dividir-para-conquistar* pode ser aplicada aos AG de diversas maneiras, tendo basicamente duas abordagens: manter uma única população ou dividir a população em várias populações locais relativamente isoladas.

Os AE, por muito tempo, focaram essencialmente na evolução de uma única população, também conhecida como população *panmictic*, caracterizada por possibilitar que todos seus indivíduos possam cruzar entre si. Entretanto, Darwin percebeu que existe uma estrutura espacial na população, e que essa estrutura pode influenciar na sua dinâmica (Mühlenbein, 1991). Em populações isoladas (*demes*) como as que vivem em ilhas, algumas espécies evoluem de maneira diferente das que vivem em ambiente não isolado. Embora a diversidade nos *demes* seja baixa, a diversidade da população geral é grande (Tomassini, 2005). Grosso (1985) *apud* Tomassini (2005) realiza um dos primeiros estudos sistemáticos sobre AGP com múltiplas populações e migração, constatando que estes exploram melhor o espaço de busca de um problema e evitam a convergência para um mínimo local, graças à sua capacidade de manter a diversidade da população geral.

Os AGP possuem maior complexidade que os AG sequenciais (Ruciński et al., 2010). No caso dos AGP com múltiplas populações, a migração é controlada por vários parâ-

metros: número de populações, topologia das conexões entre as populações, frequência da migração, número de indivíduos migrantes, seleção dos indivíduos migrantes, reposição dos indivíduos migrantes, migração síncrona ou assíncrona. Desta forma, o ajuste dos AGP é uma tarefa desafiadora.

2.3.1 Modelos de Algoritmos Genéticos Paralelos

Os AGP são classificados em 3 tipos básicos: o modelo Mestre-Escravo; o modelo *Fine-Grained* (granularidade fina) também conhecido como modelo celular; e o modelo *Coarse-Grained* (granularidade grossa) ou modelo de ilhas. Além desses, existem modelos híbridos que são combinações desses modelos (Cantú-Paz, 1998; Cantú-Paz, 2000). Luque e Alba (2011) apresentam uma classificação similar com o acréscimo do modelo de execuções independentes.

Modelo Mestre-Escravo

O modelo Mestre-Escravo (Figura 2.1) possui uma população única e global. A população é controlada pelo processo mestre, que a distribui entre os escravos para que esses realizem em paralelo a avaliação da aptidão dos indivíduos. É uma implementação paralela de AG muito comum em situações em que a aptidão de um indivíduo independe do restante da população. A comunicação ocorre somente nos momentos em que a população é enviada aos escravos e no retorno dos valores das aptidões calculadas. Esse modelo é, em geral, mais eficiente quando a função de aptidão é cara em termos computacionais, tornando o custo da comunicação entre mestre e escravos insignificante quando comparado com o custo da avaliação da aptidão.

Os algoritmos do tipo Mestre-Escravo podem trabalhar de modo síncrono ou assíncrono. Quando o algoritmo é síncrono, o mestre espera até receber a avaliação dos indivíduos de todos os escravos antes de prosseguir. Os AGP do tipo mestre-escravo síncrono exploram o espaço de busca da mesma maneira que os AG sequenciais. Desta forma, podem fazer uso direto das diretivas e teoria existente, além de apresentar resultados significativos de melhoria no desempenho. O contraponto do modelo Mestre-Escravo síncrono é que, em geral, os processos podem passar grande parte do tempo ociosos: o mestre espera que todos os escravos terminem o seu trabalho e enviem o resultado das avaliações dos indivíduos, e os escravos esperaram o mestre processar os resultados recebidos e enviar novos indivíduos para serem processados.

Quando o objetivo principal é obter melhor performance em detrimento de um comportamento igual ao dos AG convencionais, pode-se optar por utilizar o modelo Mestre-Escravo assíncrono. No modo assíncrono, o mestre gera indivíduos e os envia aos escravos para serem avaliados. Assim que os escravos terminam a sua parte e enviam os resultados para o mestre, o mestre imediatamente utiliza esses resultados para gerar uma nova população e enviá-la novamente para os escravos avaliarem. Algoritmos assíncronos têm o potencial para serem mais eficientes que os algoritmos síncronos, principalmente quando os tempos de avaliação não são constantes para todos os indivíduos. Porém, são mais complexos porque possuem parâmetros adicionais, como por exemplo, a quantidade de indivíduos gerados de cada vez, o método para incorporá-los na população. Os algoritmos assíncronos são também mais difíceis de implementar.

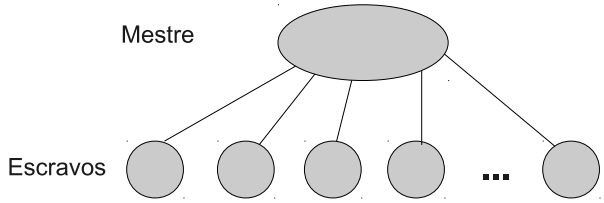


Figura 2.1: *Esquema do modelo de AGP mestre-escravo.*

Modelo *Fine-Grained*

O modelo *Fine-Grained* (Figura 2.2) possui uma população única e espacialmente estruturada. A estrutura da população é normalmente na forma de um reticulado retangular de duas dimensões e existe um indivíduo por ponto do reticulado. Se existe um processador por indivíduo, a avaliação da aptidão dos indivíduos pode ser executada simultaneamente para todos os indivíduos. A seleção e cruzamento é restrita à uma pequena vizinhança ao redor de cada indivíduo. As vizinhanças se sobrepõe e, eventualmente as características de um bom indivíduo podem se propagar pela população. Essa classe de AGP também é chamada de modelo de difusão porque a difusão de boas características através da população é análoga à difusão aleatória de partículas em alguns fluidos. Os AGP do modelo *Fine-Grained* também são conhecidos como AG celulares.

Modelo *Coarse-Grained*

O modelo *Coarse-Grained* (Figura 2.3) – também conhecido como modelo de Ilhas, modelo de Multipopulações (*multiple-demes*) e modelo Distribuído – divide a população

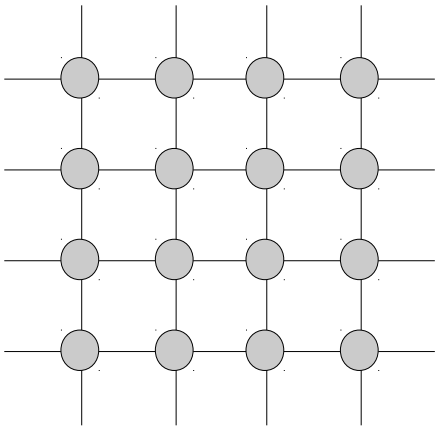


Figura 2.2: *Esquema do modelo de AGP fine-grained.*

em populações locais, ou subpopulações, que executam em paralelo o algoritmo de um AG convencional com a adição opcional de rotinas que implementam a migração de indivíduos entre as populações locais. São os modelos mais fáceis de implementar mas são os mais difíceis de se controlar, uma vez que a migração e divisão da população implicam na definição do número de subpopulações, do tamanho das subpopulações, da frequência da migração, do número de migrantes e da destinação dos migrantes, dos métodos de seleção dos migrantes e da forma como os que chegam são inseridos na população local. Os AGP desse modelo também são conhecido como Algoritmos Genéticos Distribuídos.

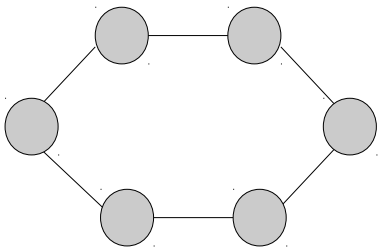


Figura 2.3: *Esquema do modelo de AGP coarse-grained.*

O esquema de comunicação entre as subpopulações mostrado na Figura 2.3 é apenas ilustrativo e mostra uma topologia em anel. A topologia da comunicação entre as subpopulações é parte da estratégia de migração dos AGP e essa topologia é mapeada sobre alguma rede física.

A estratégia de migração inclui os seguintes parâmetros:

- **Frequência da Migração.**

  A troca de indivíduos entre subpopulações ocorre a intervalos geracionais definidos ou a uma dada probabilidade para decidir a cada geração se a migração

ocorre ou não.

- **Taxa da Migração.**

  Esse parâmetro determina o número de indivíduos que participam da migração. O valor desse parâmetro pode ser dado em valor absoluto ou em valor percentual do tamanho da população.

- **Seleção dos Migrantes.**

  A forma como os indivíduos que emigram são selecionados é definida por esse parâmetro. Podem ser escolhidos os melhores ou a escolha pode ser aleatória.

- **Posicionamento dos Migrantes.**

  Esse parâmetro define como os indivíduos que migram são colocados na sua subpopulação de destino. Podem substituir os piores indivíduos ou os escolhidos de forma aleatória.

- **Topologia da Migração.**

  Esse parâmetro define a vizinhança de cada subpopulação, ou seja, as subpopulações para as quais uma dada subpopulação pode enviar (ou receber) indivíduos.

Cada subpopulação pode ter valores diferentes de parâmetros, caracterizando os chamados modelos heterogêneos.

Grosso (1985) *apud* Cantú-Paz (1998) realiza um dos primeiros estudos sistemáticos sobre os AGP com múltiplas populações e migração. Constata que os AGP exploram melhor o espaço de busca de um problema e evitam a convergência para um mínimo local, graças à sua capacidade de manter a diversidade da população geral. Grosso dividiu a população em subpopulações com migração controlada de indivíduos. Observou que características que melhoram a aptidão se espalham mais rapidamente quando as subpopulações são menores do que quando são maiores, porém quando essas subpopulações estão isoladas, a qualidade das soluções encontradas após a convergência é pior do que quando tem-se uma única população. Com baixas taxas de migração, as subpopulações se comportam como se ainda estivessem isoladas e, a qualidade das soluções se equivale à das isoladas. Com taxas de migração intermediárias, a população dividida encontrou soluções similares às das encontradas com populações únicas. Essas observações indicam a existência de uma taxa de migração crítica, abaixo da qual o desempenho dos algoritmos é prejudicado pelo isolamento das subpopulações.

Para taxas acima da taxa de migração crítica, as soluções encontradas são de mesma qualidade que as encontradas por populações *panmictic*.

A popularidade dos AG com múltiplas populações se deve à: (1) parecerem uma extensão dos AG canônicos em paralelo e realizar a troca de alguns indivíduos de tempos em tempos; (2) pouco esforço extra é necessário para converter um AG canônico em um AGP com multipopulações, uma vez que a maior parte do programa serial permanece o mesmo e algumas rotinas para implementar a migração são adicionadas; (3) os resultados da implementação de multipopulações podem ser interessantes mesmo quando a plataforma em que é executado o algoritmo não seja uma máquina paralela. Porém, os ganhos em desempenho são maiores quando o processamento é paralelo. A sua popularidade aumenta ainda mais com a atual disponibilidade de máquinas que possibilitam o processamento paralelo, como em computadores com os processadores *multicores*.

Modelos Híbridos

Os modelos híbridos combinam o modelo de multipopulações com o mestre-escravo e/ou o *fine-grained*. É também conhecido como modelo hierárquico, pois é um algoritmo de multipopulações no nível mais alto com implementações dos modelos do tipo mestre-escravo ou *fine-grained* nos níveis mais baixos.

Em (Tomassini, 2005), é apresentada uma outra classificação baseada na maneira como encontra-se estruturada a população: modelo de ilhas e modelo celular. No modelo de ilhas ou multipopulações, cada ilha é um vértice do grafo e as arestas são os caminhos por onde ocorrem os movimentos de migração. Em um nível mais baixo, a subpopulação de cada ilha pode ser vista como um grafo completo onde os vértices são os indivíduos e as arestas são as possibilidades de cruzamentos. Já no Modelo Celular, ou Modelo Difuso, os indivíduos que constituem a população estão dispostos na forma de um reticulado.

Modelo de Execuções Independentes

O modelo de Execuções Independentes consiste simplesmente na execução em paralelo do mesmo algoritmo sequencial sem interações entre as execuções independentes. Esse modelo pode ser utilizado para executar várias versões do mesmo problema com condições iniciais diferentes, e posteriormente se aplicar ferramentas estatísticas para

análise dos resultados. Quando o modelo *coarse-grained* (ver seção 2.3.1) é utilizado sem nenhuma migração e o resultado é a melhor solução dentre as soluções obtidas pelas execuções independentes, ele pode ser considerado como um modelo de execuções independentes.

2.3.2 Fundamentação Teórica

A aplicação de AGP com múltiplas populações implica na tomada de várias decisões que têm grande influência no comportamento do algoritmo. Dentre elas, tem-se a escolha da política de migração: topologia, frequência e taxa da migração, e seleção/-recolocação de migrantes. A decisão sobre os valores desses parâmetro é realizada por experimentação ou baseada nos aspectos teóricos dos AGP.

Segundo Cantú-Paz (1998), os primeiros trabalhos preocupados com a fundamentação teórica dos AGP são os trabalhos que investigam se a qualidade das soluções encontradas pelos algoritmos paralelos é melhor que a qualidade das encontradas pelos AG canônicos. Trabalhos como o de Whitley et al. (1997) mostram que a migração e a recombinação podem combinar soluções parciais em soluções melhores, favorecendo a convergência quando a função é linearmente separável (ver Definição 2.3.1). O oposto ocorre quando a função não é separável, isto é, a recombinação de soluções parciais pode resultar em indivíduos com menor aptidão. Nesse caso o AG sequencial pode levar vantagem sobre o AGP.

Definição 2.3.1 (Função Separável) Uma função $f(x)$ é separável se e somente se

$$arg \min_{x_1,\cdots,x_n} f(x_1,\cdots,x_n) = \left(arg \min_{x_1} f(x_1,\cdots), \cdots, arg \min_{x_n} f(\cdots,x_n) \right)$$

Uma função de n variáveis é separável se ela pode ser reescrita como o somatório de n funções de somente uma variável. Se uma função $f(x)$ é separável, seus parâmetros x_i são ditos independentes (Táng et al., 2010). Um exemplo de função separável é a função Rastrigin.

Definição 2.3.2 (Função Rastrigin) *A função Rastigin f_{Ras} é dada pela seguinte equação:*

$$f_{Ras}(X) = 10n + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i))$$

onde n é o número de dimensões e o mínimo global é $f_{Ras}(X^*) = 0$*, e* $X^* = [0, 0, ..., 0]$.

Em (Cantú-Paz, 2000), é apresentada uma análise teórica extensiva dos AGP e dos seus parâmetros, utilizando a hipótese dos blocos de construção (*Building Blocks*). A hipótese dos blocos de construção diz que os AG representam as possíveis soluções em esquemas ou cadeias de bits chamados de blocos de construção. As soluções são encontradas através da recombinação de bons blocos de construção que produzem blocos igualmente bons ou ainda melhores (Goldberg, 1989). Essa simplificação e as limitações impostas pela análise dificultam a utilização dos resultados apresentados em aplicações práticas (Eiben e Smith, 2011).

Outro trabalho recente com abordagem teórica na análise dos AGP é o de Lässig e Sudholt (2010), que apresenta uma análise da performance do modelo de ilhas, das fases evolucionárias independentes e a comunicação entre ilhas. Um modelo de ilha simples com migração encontra o ótimo global em tempo polinomial, enquanto populações *panmictic* e modelos de ilhas sem migração podem chegar a um tempo exponencial. Os resultados apresentados confirmam a importância da migração e contribuem para a fundamentação teórica dos algoritmos evolucionários paralelos.

2.4 Considerações do Capítulo

A bibliografia sobre Algoritmos Genéticos é extensa e os principais textos utilizados como referência neste capítulo são (Goldberg, 1989), (Michalewicz, 1996), (Lacerda e Carvalho, 2002), (Zuben, 2000).

Existem inúmeras bibliotecas de Algoritmos Genéticos de código aberto disponíveis na internet, como: GALib (`http://lancet.mit.edu/ga/`), Genesis - *GENEtic Search Implementation System* (`http://www.cs.cmu.edu/Groups/AI/areas/genetic/ga/systems/genesis/0.html`), *GA Playground - Java Genetic Algorithms Toolkit* (`http://www.aridolan.com/ga/gaa/gaa.html`), *Kanpur Genetic Algorithms Laboratory* (`http://www.iitk.ac.in/kangal/index.shtml`), dentre outros.

Em textos como os livros (Cantú-Paz, 2000), (Tomassini, 2005), (Alba, 2005), (Luque e Alba, 2011) podem ser encontradas informações mais detalhadas sobre os Algoritmos Genéticos Paralelos.

Outros trabalhos publicados recentemente e que estão relacionados a aplicações,

propostas de novos algoritmos e análises teóricas dos AGP são descritas no Apêndice A.

Capítulo 3

Processadores *Multicore*

Este capítulo mostra como a arquitetura de computadores evoluiu até a arquitetura dos processadores com múltiplos núcleos (*multicore*). Apresenta as diferentes abordagens dos projetos dos processadores *multicore* e técnicas de programação paralela.

3.1 Introdução

Desde o surgimento do primeiro computador de propósito geral há mais de 60 anos, sua tecnologia tem tido avanços impressionantes. Hoje é possível comprar por menos de 500 dólares um computador com mais desempenho, mais memória, e mais espaço de armazenamento em disco do que um computador de 1 milhão de dólares em 1985 (Hennessy e Patterson, 2006). Esse rápido avanço foi obtido pela tecnologia utilizada na construção de computadores e no projeto dos computadores.

A partir da década de 1970 surgiu o microprocessador, ou simplesmente processador. Sua capacidade de incorporar os avanços da tecnologia dos circuitos integrados e a sua produção em massa estabeleceram o seu domínio sobre o projeto de computadores. No período de 1986-2002, o crescimento do desempenho dos microprocessadores com um único núcleo atingiu as suas maiores taxas. Nos anos 1990 a importância dos multiprocessadores cresceu com os projetistas descobrindo modos de construir servidores e supercomputadores de maior desempenho do que um microprocessador sozinho e, ainda assim, usufruindo do vantajoso custo-benefício dos microprocessadores comerciais.

Segundo Hennessy e Patterson (2006), o interesse em multiprocessamento é justificado pelos seguintes fatos:

- Crescente interesse em servidores e desempenho de servidores.

- Crescimento de aplicações com grande volume de dados.
- Melhor compreensão de como usar multiprocessadores de maneira mais efetiva, especialmente nos ambientes dos servidores onde existe um paralelismo significativo.
- As vantagens de se alavancar um investimento em um projeto através da replicação em vez de projetos únicos; todos projetos de multiprocessadores possibilitam esta alavancagem.

Para entendermos o funcionamento dos processadores multinúcleo (ou *multicores*) é preciso rever os fundamentos da arquitetura básica de computadores. A próxima seção introduz os conceitos básicos sobre arquitetura de computadores e as seções seguintes descrevem os multicores, seu funcionamento e as implicações dessa nova tecnologia no desenvolvimento de software.

3.2 Arquitetura de Computadores

A arquitetura básica de um computador é organizada como um conjunto de unidades funcionais conhecido como unidade central de processamento (CPU) . Esse conjunto é constituído pelas seguintes unidades: unidade lógica e aritmética (ALU), unidade de controle, registradores e unidade de memória. A unidade lógica e aritmética é a responsável por executar efetivamente as instruções dos programas, como instruções lógicas, matemáticas, desvio de execução, etc. A unidade de controle é responsável pelo controle das ações a serem realizadas pelo computador, comandando todos os outros componentes. Os registradores são pequenas memórias velozes que armazenam instruções ou valores que são utilizados no controle e processamento de cada instrução. A unidade de memória é um dispositivo de hardware que transforma endereços virtuais em endereços físicos e administra a memória principal do computador.

Termos como CPU, *chip*, processador, microprocessador, núcleo, *core* e elemento processador são frequentemente usados para identificar a mesma entidade física, dependendo do contexto em que aparecem. O processador corresponde à unidade de processamento central (CPU) e é manufaturado na forma de um circuito integrado ou *chip*, que por sua vez é conhecido como microprocessador. Logo, CPU, processador, *chip* e microprocessador podem se referir à mesma entidade física dependendo do contexto. Núcleo (ou *core* em inglês) e elemento de processamento referem-se a

uma unidade de processamento local. Um processador pode conter um ou mais núcleos. *Multicore* é o processador com múltiplos núcleos, e *monocore* ou *singlecore* é o processador com apenas um núcleo.

O processo de fabricação de um microprocessador agrega todos os seus componentes em um pacote que é encaixado no computador através de um *socket*. *Socket* é o encaixe físico existente na placa-mãe para receber um *chip*. Nos computadores *desktop*, a placa-mãe, em geral, possui apenas um *socket*, enquanto que servidores podem ter múltiplos *sockets*.

3.2.1 Computadores Paralelos

A ideia de se usar vários processadores para melhorar o desempenho existe há muito tempo. Cerca de 40 anos atrás, Flynn definiu uma maneira de categorizar os computadores a partir de duas dimensões diferentes. Uma dimensão é em relação aos fluxos de instruções que um computador de uma arquitetura pode executar ao mesmo tempo. A outra é em relação aos fluxos de dados que podem ser processados simultaneamente. A taxonomia de Flynn consiste em quatro categorias:

- *Single instruction stream, single data stream* (SISD) - Existe um único processador que tem acesso a um programa e espaço de armazenamento. É a categoria do monoprocessador com arquitetura sequencial convencional, também conhecido como modelo de arquitetura *von Neumann*;

- *Single instruction stream, multiple data stream* (SIMD) - A mesma instrução é executada por múltiplos processadores usando diferentes fluxos de dados. Computadores SIMD exploram o chamado paralelismo de dados através da aplicação da mesma operação a múltiplos dados em paralelo. Cada processador possui sua própria memória mas existe uma única memória de instruções e processador de controle, que busca e despacha instruções. Exemplos de aplicações em que o modelo SIMD pode ser muito eficiente são as aplicações com alto grau de paralelismo nos dados, como aplicações multimídia e aplicações gráficas;

- *Multiple instruction stream, single data stream* (MISD) - Vários processadores executam múltiplas operações sobre os mesmos dados compartilhados. É considerada uma categoria hipotética, e exemplos de sistemas pertencentes a essa categoria são os *systolic arrays*, as aplicações de filtros de frequência múltipla

operando sobre um único fluxo de sinal e aplicações de algoritmos múltiplos de criptografia tentando quebrar uma mensagem codificada única;

- *Multiple instruction stream, multiple data stream* (MIMD) - Múltiplos processadores onde cada processador executa seu próprio conjunto de instruções e opera sobre seus próprios dados. Computadores MIMD exploram o chamado paralelismo de *thread*, uma vez que múltiplos *threads* operam em paralelo. Em geral, o paralelismo a nível de *thread* é mais flexível do que o paralelismo a nível de dados. Aplicações com tarefas independentes e assíncronas podem se beneficiar desse modelo.

Arquiteturas mais recentes que não se relacionam diretamente com as categorias de *Flyn* são representadas na classificação de *Duncan* (Duncan, 1990), que divide as arquiteturas em síncronas e assíncronas.

Uma extensão da taxonomia de *Flyn* é apresentado em (Alba, 2005) e mostrada na Figura 3.1, onde a classe MIMD é dividida em multiprocessadores nos quais todos os processadores tem acesso direto a todas as memórias e em multicomputadores onde a memória é distribuída. Os multiprocessadores são classificados de acordo com o tempo de acesso à memória ser uniforme (*uniform memory access* - UMA) ou não (*non-uniform memory access* - NUMA). A classe UMA é dividida entre o tipo de interconexão entre processadores, que pode ser por barramento (*bus*) ou ponto-a-ponto *switched*). A classe NUMA possui uma distinção entre os que possuem mecanismo para manter a *cache* coerente (CC-NUMA) ou não (NC-NUMA). Os multicomputadores são sistemas distribuídos compostos de vários computadores interligados através de uma rede. Esses computadores podem ser PCs e *workstations* interligados por uma rede de comunicação de propósito geral como uma *Fast Ethernet*, *Gigabit Ethernet* ou *Myrinet*, formando um *cluster* (*cluster of workstations* - COW). Os sistemas da classe MPP (*massive parallel processor*) são compostos por centenas de processadores que podem estar geograficamente distantes e interligados pela internet (*grids*), ou agrupados em um computador.

Os modelos apresentados pela taxonomia de *Flynn* são modelos macros, uma vez que alguns multiprocessadores são híbridos dessas categorias. A maioria dos multiprocessadores adota o modelo MIMD que explora o paralelismo de *threads*, oferecem maior flexibilidade e podem ser construídos aproveitando-se as vantagens dos micropro-

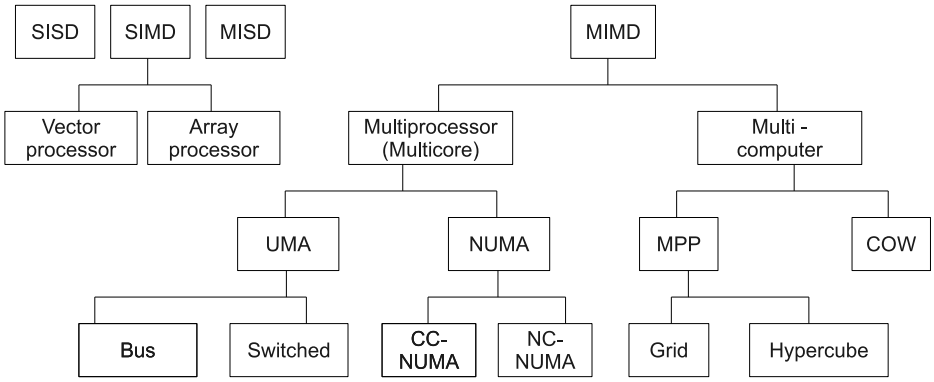


Figura 3.1: *Extensão da taxonomia de Flyn. Fonte: (Alba, 2005)*

cessadores comerciais. Uma classe popular de computadores MIMD são os *clusters* que usam microprocessadores e redes comerciais, desta forma alavancando a tecnologia comercial o máximo possível. Em (Dongarra et al., 2005), o autor discute a terminologia e taxonomia dos sistemas paralelos e a distinção entre diferentes *clusters*.

3.3 Processadores *Multicore*

Na década de 1990, o aumento da capacidade de um único circuito integrado (*chip*) permitiu que projetistas colocassem múltiplos processadores em um único encapsulamento. Essa tecnologia foi chamada de *on-chip multiprocessing* ou *single-chip multiprocessing* e passou a ser conhecida como *multicore*. O processador *multicore*, ou simplesmente *multicore*, é um projeto em que um único processador físico contém mais de um núcleo lógico de um processador. Dentro do pacote de um processador multinúcleo, outra forma de se referir ao processador *multicore*, encontram-se todos os circuitos e a lógica de dois ou mais processadores. No projeto de um *multicore*, os múltiplos núcleos compartilham alguns recursos, como memória *cache* e barramentos de I/O. Dessa forma é possível que um sistema possa executar mais tarefas simultaneamente e obter um desempenho global maior.

Usar múltiplas cópias de um microprocessador em um multiprocessador alavanca o investimento no seu projeto através da replicação. Assim como é vantajoso usar vários microprocessadores na construção de um multiprocessador, o projeto de microprocessadores *multicore* usa a replicação para alavancar o investimento no seu projeto e torna-se mais vantajoso do que o projeto de um novo microprocessador mononúcleo superescalar que enfrenta problemas de difícil solução, como é descrito a seguir.

O ritmo acelerado do crescimento do desempenho dos processadores apresenta uma

diminuição a partir de 2002. Essa desaceleração foi causada pelos seguintes problemas (Asanovic et al., 2006):

- Barreira do Consumo de Energia (*Power Wall*): a dissipação de calor e energia limita o incremento no desempenho dos processadores;
- Barreira da memória (*Memory Wall*): o acesso e armazenamento de dados na memória principal é lento, pode levar em torno de 200 *ciclos* do *clock* enquanto que uma operação aritmética de multiplicação leva apenas 4 ciclos do *clock*. A eficiência na transferência de dados da memória principal para o processador (e vice-versa) limita o desempenho dos programas;
- Barreira do paralelismo em nível de instrução (*ILP Wall*): o paralelismo na execução das instruções chegou ao seu limite.

Em 2004 a Intel cancelou o projeto de um processador de alto desempenho com um único núcleo e uniu-se à IBM e Sun na adoção de múltiplos processadores por *chip* (CMP - *chip multiprocessor*), os chamados processadores *multicore* (Patterson, 2010). Com os microprocessadores *multicore*, o usuário programador precisa escrever o seu código de forma paralela para que sua aplicação maximize o ganho de desempenho com a atualização do processador.

3.3.1 Multithreading

A execução de programas corresponde a execução de sequências de instruções chamadas *threads*. Um programa sequencial consiste de uma única *thread*. Os sistemas operacionais antigos eram capazes de executar apenas uma tarefa de cada vez. Depois evoluíram para multitarefa (*multitasking*), onde um programa era suspenso por um tempo enquanto outro era executado. A troca rápida de programas em execução proporcionava a impressão de que os programas estavam sendo executados simultaneamente, porém o processador estava, na realidade, executando somente uma *thread* de cada vez.

Com a evolução os processadores passaram a executar várias instruções em paralelo, ou seja, várias *threads* simultaneamente (*simultaneous multithreading* - SMT) . A Intel batizou essa tecnologia que possibilita que um núcleo execute duas *threads* de *Hyper-Threading* (HT). Agora, dois programas podem ser executados de forma simultânea sem realizar a troca de contexto entre tarefas (*multitasking*). Para o sistema

operacional, é como se existissem dois processadores. Essa capacidade de SMT é limitada pela disponibilidade dos recursos compartilhados pelas duas *threads*, ou seja, o SMT não consegue atingir o mesmo desempenho obtido por dois processadores distintos executando duas *threads*. Dois núcleos apresentam melhor desempenho do que um único núcleo com SMT, pois cada um possui seu próprio conjunto de recursos (Akhter e Roberts, 2006).

O projeto dos processadores *multicore* permite que cada núcleo possua sua própria *pipeline* fila de execução e cada núcleo possui recursos suficientes para executar instruções sem bloquear os recursos de outras *threads*. Cada núcleo executa de forma mais lenta (menor frequência de *clock*) e dissipa menos calor do que um processador com um único núcleo. A capacidade de execução em paralelo dos vários núcleos, quando somada, é maior do que a de um processador com um único núcleo. A Figura 3.2 apresenta uma comparação entre diferentes arquiteturas de processadores.

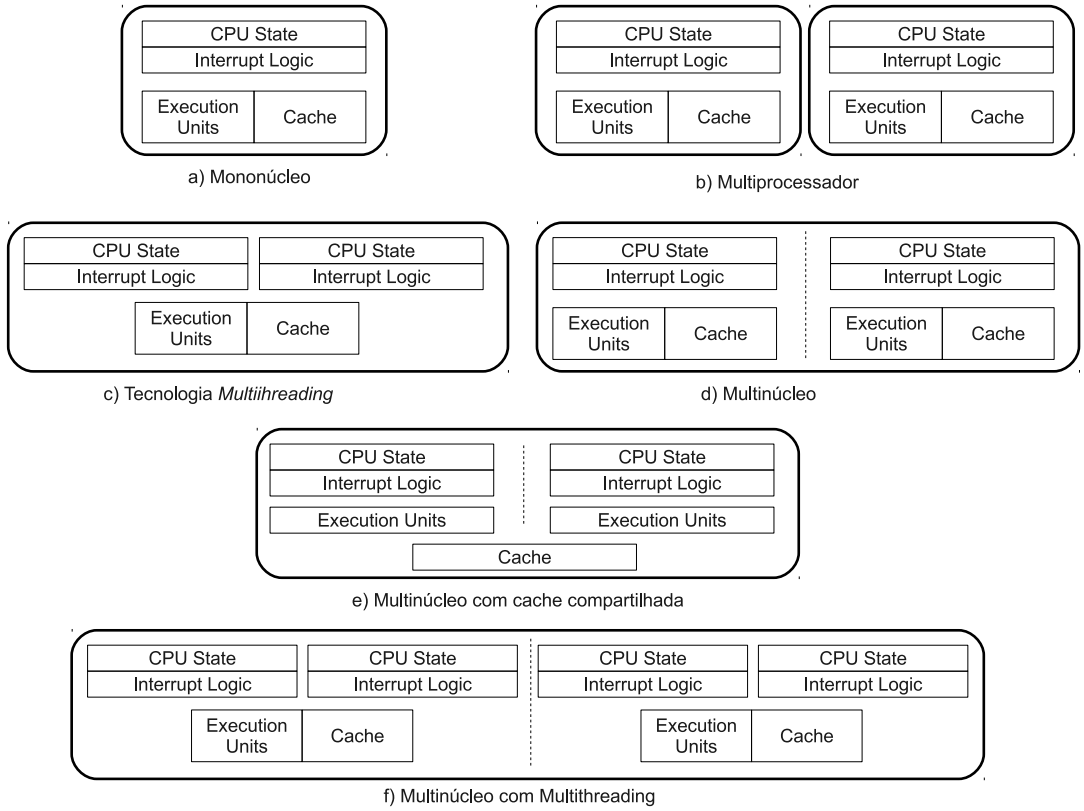


Figura 3.2: *Comparação entre arquiteturas de processadores. Fonte: (Akhter e Roberts, 2006)*

De maneira geral, os *multicores* beneficiam todos os tipos de programas mas em especial programas que fazem uso do *multithreading*, ou seja, programas paralelos. Na seção 3.5 são apresentados dois padrões de programação paralela.

3.3.2 Organização da Memória

De acordo com Hennessy e Patterson (2006), a maioria dos computadores paralelos são baseados no modelo MIMD. Uma classificação dos computadores MIMD também pode ser feita de acordo com a organização da memória: memória compartilhada e memória distribuída. Essa classificação pode ser realizada a partir do ponto de vista do programador ou da organização física da memória. Esses pontos de vista podem não coincidir: é possível existir uma memória compartilhada virtual no topo de uma memória fisicamente distribuída, segundo Rauber e Rünger (2010).

Nos modelos de memória compartilhada, os processadores compartilham uma memória central principal e possuem uma relação simétrica e tempo de acesso uniforme a partir de qualquer processador. Muitas vezes os que seguem esse modelo são chamados de multiprocessadores simétricos (SMP - *symmetric shared memory multiprocessors*) e esse estilo de organização de memória é chamado de UMA (*uniform memory access*) pois todos os processadores possuem latência de memória uniforme. O compartilhamento de memória torna-se menos atrativo à medida que aumenta o número de processadores. A Figura 3.3 mostra o esquema desse modelo.

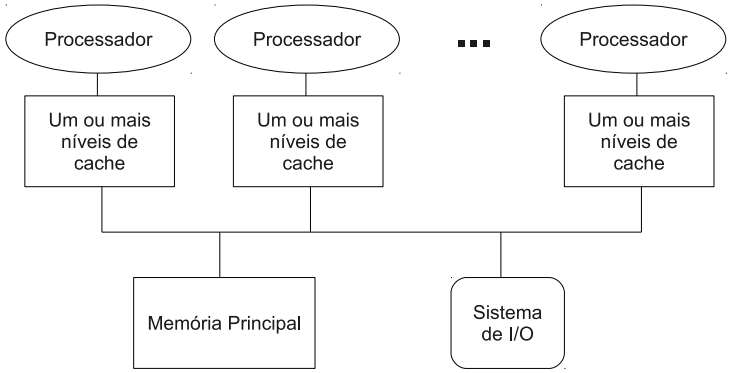


Figura 3.3: *Estrutura básica da arquitetura com memória compartilhada. Fonte: (Hennessy e Patterson, 2006).*

A arquitetura com memória fisicamente distribuída, como mostra a Figura 3.4, consegue atender a um número maior de processadores do que a memória compartilhada. A memória é distribuída entre os processadores e existe um sistema de conexão. Apesar dos benefícios de baixa latência de acesso à memória local, a comunicação entre processadores é mais complexa. Existem dois métodos alternativos para a organização do espaço de endereçamento.

O primeiro método consiste em usar um espaço de endereçamento compartilhado

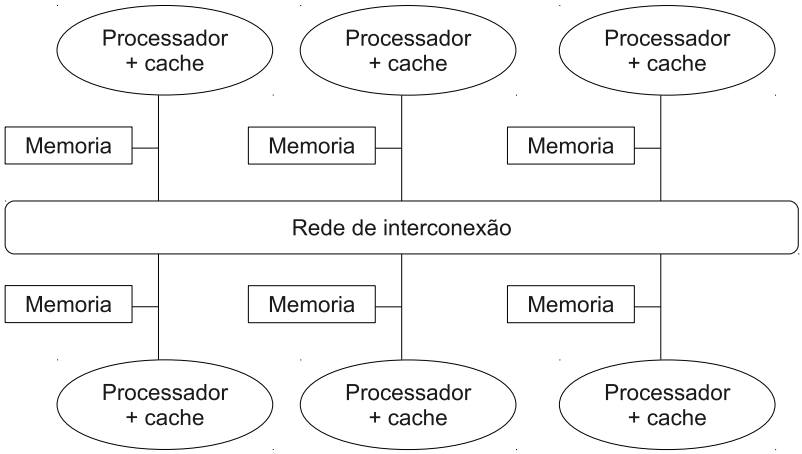


Figura 3.4: *Esquema de arquitetura com memória distribuída. Fonte: (Hennessy e Patterson, 2006).*

para comunicação como acontece com arquiteturas de memória compartilhada. As memórias fisicamente separadas podem ser endereçadas como um espaço de endereçamento logicamente compartilhado. Essa arquitetura é conhecida como memória compartilhada distribuída (DSM - *distributed shared-memory*). O espaço de memória compartilhada não é um espaço único centralizado como nos SMPs. Nos DSMs o tempo de acesso à memória depende da localização do dado na memória, e por isso essas arquiteturas são conhecidas como NUMAs (*nonuniform memory access*).

Outra maneira é ter o espaço de endereçamento formado por múltiplos espaços de endereçamento privados, que são logicamente disjuntos e não podem ser endereçados por um processador remoto. Dessa forma, o mesmo endereço físico em dois processadores diferentes referem-se a dois locais diferentes.

Para cada método de organização do espaço de endereçamento existe um mecanismo de comunicação. No caso do compartilhamento de memória, a comunicação acontece através de operações de leitura e escrita no espaço compartilhado. Quando o espaço de endereçamento é distribuído, a comunicação ocorre de forma explícita com a troca de mensagens entre processadores.

3.3.3 Memória *Cache*

Devido à grande diferença entre a velocidade dos processadores e memória principal, memórias *cache* foram incorporadas aos processadores. Estas *caches* são pequenas mas de alta velocidade que servem como *buffers* temporários para as memórias principais, maiores e mais lentas. As memórias *cache* são um recurso de hardware para melhorar o desempenho, pois elas podem reduzir o tempo de latência de acesso a dados de uma

ordem de grandeza quando comparado com o acesso à memória principal.

Memórias rápidas são recursos caros e por isso a memória *cache* é organizada em níveis hierárquicos: um nível é sempre menor, mais rápido e mais caro que o inferior. A Figura 3.5 mostra o esquema de hierarquia em vários níveis de memória. O objetivo é estabelecer um sistema de memória com custo por *byte* quase tão baixo quanto o do nível mais barato de memória e velocidade tão rápida quando o nível mais caro.

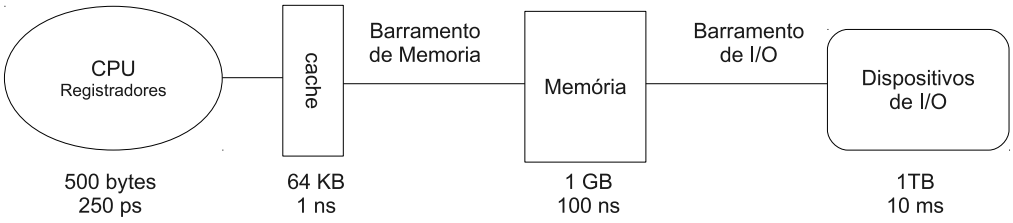


Figura 3.5: *Níveis em uma hierarquia de memória com valores típicos de tamanho e velocidade. Fonte: (Hennessy e Patterson, 2006)*

No processador IBM Power5 o acesso a dados localizados na *cache* L2 requer 14 ciclos de processador enquanto que acessar dados localizados na memória principal requer 280 ciclos.

Nos processadores atuais, existem 2 ou 3 níveis de *cache*. A *cache* nível 1 (L1) é a mais próxima do núcleo de execução do processador e possui o tempo de acesso mais curto, porém menor capacidade de armazenamento. A *cache* nível 2 (L2) é a seguinte na hierarquia e possui tempos de acesso mais longos e maior capacidade de armazenamento. Finalmente, tem-se a *cache* nível 3 (L3) com tempos de acesso mais longos ainda e maior capacidade de armazenamento. Nos *multicores*, a *cache* do último nível, que é o último nível antes de se acessar a memória principal, é geralmente compartilhada entre vários núcleos. Esse componente é, na maioria das vezes, a *cache* L2 ou L3.

Quando um dado compartilhado é armazenado na *cache*, o valor do dado deve ser replicado nas várias *caches* para que o tempo de acesso a ele seja reduzido em todos os processadores e que ele possa ser lido simultaneamente por todos os processadores. Esse mecanismo traz um novo problema: manter a coerência da *cache*, ou seja, se múltiplas cópias do mesmo dado podem existir em diferentes *caches* simultaneamente, e se os processadores estão habilitados a atualizar livremente suas próprias cópias, uma visão de memória inconsistente poderá ocorrer.

Segundo Blake et al. (2009), ao contrário das arquiteturas com único núcleo, nos

multicores as *caches* fazem parte de um conjunto maior e mais complexo: o sistema de memória. Além das *caches*, o sistema de memória dos *multicores* tem o modelo de consistência dos dados, os mecanismos para a coerência entre *caches* e a forma de interconexão entre elas. Esses componentes determinam como os núcleos se comunicam, impactando na programação e no desempenho das aplicações paralelas, e no número de núcleos que o sistema admite.

3.3.4 Consistência dos Dados

O modelo de consistência define como as operações na memória podem ser reordenadas enquanto o código está sendo executado. Ele determina o quanto de esforço é requerido do programador ao escrever códigos paralelos. Modelos mais fracos exigem que o programador defina explicitamente como o código deve ser agendado no processador e possui protocolos de sincronização mais complexos. Já os modelos mais robustos requerem menos esforço de programação por parte do usuário com relação às sincronizações, porém, além de complexos e difíceis de serem projetados, são mais lentos.

Caches podem ser marcadas e gerenciadas automaticamente pelo hardware ou explicitamente gerenciadas pela memória local. As *caches* marcadas e automaticamente gerenciadas são as mais comuns, pois são transparentes para o fluxo de instruções que acredita ter acesso à uma memória uniforme. A principal desvantagem das *caches* automaticamente gerenciadas é que o seu tempo de acesso é não determinístico e usam a área do circuito a que pertencem para armazenar marcações para cada entrada. Contrariamente, armazenamentos locais proveem tempo de acesso determinístico, pois são explicitamente gerenciados pelo fluxo do software sendo executado e proporcionam mais espaço de armazenamento para a mesma área de memória porque não precisam de marcações. Esse gerenciamento de software pode ser muito oneroso e, na maioria dos casos, é somente opção para aplicações que possuem grandes requisitos de desempenho em tempo real.

O tamanho que os diferentes níveis de *cache* devem assumir depende muito da aplicação. *Caches* maiores oferecem melhor desempenho até certo ponto. A partir de um determinado tamanho, seu desempenho começa a declinar. *Caches* grandes também podem não contribuir para o desempenho de aplicações que usam os dados apenas uma vez, tal como a decodificação de vídeo. Nesse caso é desejável que a *cache*

trabalhe com modos de operação diferentes: para acessos em fluxo (*streaming*) e para acessos com comportamento normal. O Microsoft Xeon possui essa funcionalidade para prevenir que a *cache* seja preenchida com dados que não serão reutilizados. As *caches* são dimensionadas para ocupar o máximo da área do circuito do qual faz parte e consumir o máximo de energia permitido. Essa é uma tendência nas arquiteturas de propósito geral com grande reuso de dados, como o Intel Core i7.

3.3.5 Interconexão e Coerência

A interconexão entre os diferentes níveis na hierarquia da memória é responsável pela comunicação geral entre os elementos de processamento e também pela coerência, quando essa estiver presente, entre *caches*. Essa interconexão pode ser feita de várias maneiras: por meio de um barramento, de um anel, ponto-a-ponto, ou mesmo através do conceito de *network-on-chip* (NoC). Cada um dos tipos de interconexão tem suas vantagens e desvantagens. O barramento é o projeto mais simples, porém quando conecta um número grande de núcleos, ele torna-se inviável tanto em termos de latência quanto de largura de banda. O anel possui o mecanismo de roteamento mais simples, mas tem limitações quanto a largura da banda. O ponto-a-ponto tem boa latência, mas pode tornar-se inviável, pois o número de ligações necessárias cresce exponencialmente com o aumento do número de elementos de processamento. O NoC tem boa escalabilidade, mas é de difícil implementação.

Os mecanismos para manter a coerência dos dados são fundamentais na definição do modelo de programação que deve ser usado na arquitetura. Através deles, uma única imagem da memória é mantida automaticamente para o acesso de todos os processadores. Portanto, esses mecanismos são essenciais entre aplicações que compartilham informação. Eles são muito comuns em processadores de propósito geral como o ARM Cortex A9 (ARM Ltd., September, 2009). Dois modelos de coerência podem ser implementados: baseado em *broadcast* e baseado em *diretório*.

O modelo de coerência baseado em *broadcast* é simples: a coerência é mantida pelo aviso a todos os elementos de processamento de que um determinado elemento precisa acessar uma informação de uso global. Esses avisos são tratados um por vez e os conflitos são resolvidos através de sua sequencialização. De uma forma geral, essa abordagem é aplicável a um número pequeno de processadores. Em geral, são suportados da ordem de oito núcleos, como no caso do Intel Core i7.

A coerência baseada em *diretório* permite que um grande número de elementos de processamento sejam interligados uma vez que múltiplas ações de coerência podem ser tomadas simultaneamente. Um diretório é formado por nós e possui informações sobre quais *caches* contém quais endereços de memória. Cada endereço é atribuído a um nó do diretório. Quando é feito um acesso, o processador consulta o nó em que o endereço está armazenado para saber quais processadores estão fazendo uso daquela informação. Esse processador pode negociar com os demais para obter as permissões de acesso que ele deseja. A coerência baseada em *diretório* é mais indicada para modelos de consistência fracos e sistemas com muitos núcleos, como o Tilera TILE 64.

Algumas arquiteturas *multicore* omitem os mecanismos de coerência entre *caches* para reduzir a complexidade, como o IBM Cell. Essa ausência implica que a coerência dos dados deve ser garantida pelo software durante a sua execução. Isso limita o modelo de programação a variantes do modelo de passagem de mensagens (ver seção 3.5.1). Para domínios de aplicação que compartilham pouca informação, essa pode ser uma boa opção.

3.4 Modelos *Multicore* Comerciais

A Tabela 3.1 mostra uma seleção de *multicores* comerciais. Os quatro primeiros são *multicores* de propósito geral. A microarquitetura dos seus núcleos é tradicional e baseada em um mononúcleo convencional e poderoso. Todos empregam um número modesto de cópias idênticas desses núcleos com *caches* grandes. São arquiteturas direcionadas para o mercado de *desktops* e de servidores onde o baixo consumo de energia não é um fator prioritário. As demais linha da Tabela 3.1 apresentam arquiteturas direcionadas para o mercado de computação móvel e embarcada. Elas apresentam núcleos de propósito geral bastante apropriados para aplicações dominadas por estruturas de controle. Nesse contexto, o consumo de energia é muito importante pelo fato de que eles serão alimentados por baterias.

O Intel Core i7 é um processador de propósito geral em todos os aspectos, feito com o objetivo de fazer tudo bem feito. Isso é obtido ao preço de alto consumo de energia que pode chegar a 130W. Pode ser implementado com até 8 núcleos, cada um despachando até quatro instruções *fora de ordem* simultaneamente. Além disso, ele faz uso do conceito de *symmetric multithreading*, conforme pode ser visto na Figura 3.6. Esses núcleos contém uma série de aprimoramentos complexos para que possam extrair

Proc.	ISA	Arquitetura	#	Cache	Coer.	Intercon.	Cons. Max	Freq.
AMD Phenon	x86	Tres-vias fora-de-ordem 128-B SIMD	4	64 KB IL1 e DL1 256 KB L2 2-6 MB L3 shared	Dir.	Ponto a Ponto	140 W	2,5GHz a 3,0GHz
Intel Core i7	x86	Quatro-vias fora-de-ordem Duas-vias SMT 128-B SIMD	2 a 8	32 KB L1 IL1 e DL1 256 KB L2 8 MB L3 shared	Bcast	Ponto a Ponto	130 W	2,66GHz a 3,33GHz
Sun Niagara	sparc	Duas-vias em-ordem Oito-vias SMT	8	16 KB IL1 e 8 KB DL1 4 MB L2 shared	Dir.	Comut.	60-123W	900MHz a 1,4GHz
Intel Atom	x86	Duas-vias em-ordem Oito-vias SMT 128-B SIMD	1 a 2	32 KB IL1 e DL1 512 KB L2 shared	Bcast	Barr.	2-8 W	800MHz a 1,6GHz
ARM Cortex-A9	arm	Tres-vias fora-de-ordem	1 a 4	(16, 32, 64) KB IL1 e DL1 Até 2 MB L2	Bcast	Barr.	1 W (s/ *cache*)	n/a
XMOS XS1-G4	xcore	Uma-via em-ordem Oito-vias SMT	4	64 KBLCL Store/Core	none	Comut.	0,2 W	400 Mhz

Tabela 3.1: *Processadores multicore de propósito geral. Fonte: (Blake et al., 2009).*

o máximo de paralelismo possível na execução de uma única *thread*. Eles também possuem uma unidade com vetor SIMD de 128-bits pra obter as vantagens de eventuais paralelismos sobre os dados. Para continuar compatível com a maioria dos processadores Intel, seu conjunto de instruções é baseado no CISC x86. A concepção do Intel Core i7 permite que ele faça muitas coisas bem, mas arquiteturas especializadas em determinados domínios de aplicação podem obter desempenho equivalente ou melhor, e consumindo menos energia. O sistema de memória é típico de arquiteturas *multicore* de propósito geral com poucos núcleos. As *caches* são grandes e há total coerência entre elas. A coerência é do tipo *broadcast*, que é suficiente para o número de núcleos projetados. Esse conjunto de características reunidas dá origem a um processador que é bom para uma grande variedade de aplicações, desde que o consumo de energia não seja um problema.

Outra inovação introduzida pelo Intel Core i7 codinome *Nehalem* é o recurso chamado de *Turbo Boost* que dinamicamente varia a frequência dos núcleos do processador. A frequência é determinada pela temperatura do núcleo, o número de núcleos ativos, a potência estimada e a o consumo de energia atual estimado (Charles et al., 2009).

3.5 Programação Paralela

A programação paralela é baseada no conceito de divisão e conquista, onde uma tarefa, que é complexa ou exige grande capacidade de processamento, é dividida em tarefas menores e independentes.

Um modelo de programação é uma ponte entre o modelo de uma aplicação (alto

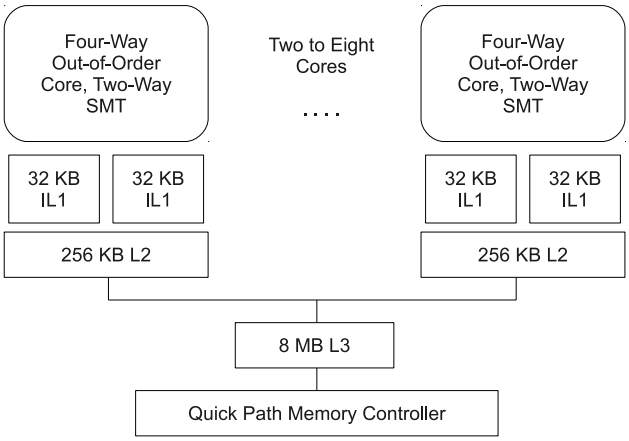


Figura 3.6: *Diagrama de blocos do Multicore Intel Core i7 (Blake et al., 2009)*

nível) e a sua implementação no hardware disponível (baixo nível). Tais modelos permitem ao programador abstrair os detalhes da implementação de baixo-nível e se concentrar no desenvolvimento do software, mantendo um equilíbrio entre a produtividade e a eficiência da implementação. A eficiência da implementação é sempre importante quando se está paralelizando uma aplicação, uma vez que programas com poucos requisitos de desempenho podem ser executados sequencialmente. Em (Asanovic et al., 2006), são apresentados dois objetivos fundamentais para se manter esse equilíbrio:

- A opacidade abstrai os detalhes de baixo nível e livra o programador de ter que conhecer os complicados detalhes da arquitetura e aumenta a sua produtividade.
- A visibilidade faz com que os elementos importantes do hardware sejam visíveis ao programador, permitindo que ele consiga atender aos requisitos de desempenho da aplicação através da manipulação desses elementos, como a localização dos dados, limites de *threads*, etc.

Computadores paralelos possuem uma arquitetura básica formada por: processadores, memória e rede de comunicação. A memória pode estar organizada de forma distribuída (cada processador possui a sua própria memória) ou compartilhada, e essa organização tem papel importante na decisão de qual modelo de programação paralela utilizar. Sistemas que proveem um espaço de memória uniformemente compartilhada são muito populares entre programadores. Entretanto, o dimensionamento destes para sistemas maiores é um desafio. A manutenção da consistência da memória também fica ainda mais difícil quando vários núcleos podem atualizar as mesmas localizações de memória. Sistemas com memória explicitamente dividida evitam esses problemas mas os programadores tem que lidar com detalhes de baixo nível.

Tarefas que são executadas simultaneamente adicionam detalhes que não aparecem na computação sequencial. A dependência de um mesmo recurso por várias tarefas que são executadas em paralelo é um exemplo onde é necessário maior atenção e cuidado. É necessário ainda o entendimento do equilíbrio entre as partes do computador paralelo e sua influência no desempenho. Fatores como a velocidade do processador, a comunicação, e a hierarquia de memória afetam o desempenho e a portabilidade do código.

Na década de 1990, dois padrões de programação dominaram a computação paralela: o *Message Passing Interface* (MPI) e *Open Multiprocessing* (OpenMP). Além desses, havia também os padrões PVM (Parallel Virtual Machine) e HPF (High Performance Fortran). MPI e OpenMP ajudaram o desenvolvimento de aplicações paralelas ao prover padrões abertos e portáveis para várias tecnologias proprietárias. Ainda assim, MPI e OpenMP requerem um conhecimento aprofundado da computação paralela, uma curva de aprendizagem grande para a maioria dos programadores (Kim e Bond, 2009).

3.5.1 Message-Passing Interface (MPI)

O *Message Passing Interface* (MPI) é uma especificação para desenvolvedores e usuários para a normatização das bibliotecas de comunicação de dados entre muitos computadores através da troca de mensagens. Este padrão foi criado nos anos 1990 e é baseado em convenções da MPI Forum[1], que reúne fornecedores, pesquisadores, desenvolvedores e usuários. É considerado um padrão industrial para o desenvolvimento de programas e é suportado em virtualmente todas as plataformas computacionais. Diversas implementações estão disponíveis, livres ou proprietárias, como por exemplo: MPICH, OpenMPI, MVAPICH, MS MPI, Intel MPI. O MPI Oferece suporte para implementações em ANSI C, ANSI C^{++}, ANSI Fortran (Fortran90), Java e R.

Uma aplicação implementada com MPI é constituída por processos que se comunicam e que acionam funções para o envio e recebimento de mensagens entre eles. Geralmente, um número fixo de processos é criado inicialmente. Cada um desses processos pode executar diferentes programas. MPI suporta tanto a comunicação ponto-a-ponto quanto a coletiva, ou seja, os processadores podem efetuar operações para enviar mensagens de um determinado processo a outro, ou um grupo de processos pode efetuar operações de comunicações globais. É capaz de suportar comunicação assíncrona e pro-

[1] www.mpi-forum.org

gramação modular, através de mecanismos de comunicadores que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna. As principais vantagens do MPI são: portabilidade, disponibilidade, funcionalidade e padronização.

Existem duas versões do MPI: MPI-1, que dá ênfase a passagem de mensagem e tem um ambiente estático de execução; e o MPI-2, que inclui novas características, tais como: I/O paralelizável, gerenciamento dinâmico de processos e operações de leitura e escrita em memória remota. MPI-2 foi criado de modo que o MPI-1 seja um subconjunto do MPI-2, de modo que programas em MPI-1 continuam funcionando em implementações de MPI-2. O principal destaque na especificação de MPI-2 é o gerenciamento dinâmico dos processos. Esse gerenciamento permite a criação e a finalização cooperativa de processos mesmo depois da aplicação ter começado sua execução, com garantia da integridade do sistema. Permite, também, estabelecer comunicação entre processos que foram disparados separadamente, e existem operações coletivas e de sincronização, convenientes em ambientes de memória compartilhada.

A implementação MPICH2 (MPICH, 2011) é livre e possui *benchmarks* onde foram testadas e comprovadas a sua escalabilidade nos processadores *multicore* (Balaji et al., 2011) e o seu desempenho na troca de mensagens em memória compartilhada (Buntinas et al., 2007).

O modelo de comunicação por troca de mensagens é definido como um conjunto de processos que possuem acesso à memória local e a comunicação entre os processos é baseada no envio e recebimento de mensagens. A transferência de dados entre processos requer operações de cooperação entre cada processo (uma operação de envio deve estar associada a uma operação de recebimento).

Neste trabalho, foi utilizado o MPI, especificamente o MPICH2, na implementação do AGP.

3.5.2 Open Multi-Processing (OpenMP)

Open Multi-Processing (OpenMP[2]) é uma API desenvolvida para sistemas que utilizam memória compartilhada, com suporte às linguagens C, C^{++} e Fortran, com em diferentes arquiteturas, como Linux, Unix e Windows. Além de portabilidade, tem como objetivos apresentar um modelo escalável e fornecer ao programador uma interface simples e flexível.

[2]www.open-mpi.org

Por meio de *threads* que são criadas a partir de uma única *thread* principal, o programador pode manipular como será executado seu código, especificando quais os trechos que devem ser paralelizados e quais variáveis deverão ser compartilhadas. Tais informações são fornecidas através de diretivas para o compilador, semelhantes a comentários. As principais vantagens de OpenMP, em relação aos outros modelos, residem na despreocupação com a sincronização de mensagens, na flexibilidade em relação às aplicações sequenciais (um compilador comum interpretará as diretivas OpenMP apenas como comentários) e na possibilidade de paralelizar diversos trechos de código sem modificá-los estruturalmente (diminuindo a incidência de erros de programação).

Porém, a interface ainda não apresenta alguns recursos desejáveis, como um verificador de erros confiável, garantia de máxima eficiência no uso da memória compartilhada e sincronização automática de entrada e saída de dados. Atualmente encontra-se na versão 3.0 (2008) e é administrada pela comunidade OpenMP ARB (*OpenMP Architecture Review Board*), composta por diversas empresas, como: AMD, IBM, Sun Microsystems e Intel.

3.6 Considerações do Capítulo

As implicações da adoção da tecnologia *multicore* como estratégia de evolução dos processadores trouxe implicações em diversas áreas, tais como, a programação e desenvolvimento de sistemas, nos sistemas operacionais e no currículo dos cursos de formação de profissionais que atuam nessas áreas. No Anexo B é apresentado um levantamento de trabalhos com temas relacionados às implicações mencionadas.

O livro de Hennessy e Patterson (2006) é abrangente sobre arquitetura de computadores, com detalhada descrição da arquitetura dos processadores. As técnicas de programação paralela, importantes para o desenvolvimento de programas eficientes para processadores *multicore*, são o foco de Rauber e Rünger (2010).

Capítulo 4

Planejamento e Análise Estatística de Experimentos

O objetivo deste Capítulo é introduzir os conceitos empregados neste trabalho. São conceitos relacionados ao planejamento e à análise estatística de experimentos.

4.1 Introdução

A experimentação tem sido usada em diversas áreas do conhecimento. O planejamento experimental estatístico tem início com o trabalho pioneiro de R. A. Fisher entre 1920 e início de 1930 (Steinberg e Hunter, 1984). Seu trabalho teve grande influência no uso de estatística na agricultura e nas ciências da vida. Nos anos 1930, surgiram as aplicações do planejamento experimental na indústria e a compreensão de que os experimentos na indústria são diferentes dos experimentos na agricultura: a variável de resposta pode, em geral, ser observada em menor tempo, e o experimentador pode obter informações cruciais a partir de um pequeno conjunto de execuções que podem ser úteis no planejamento de um próximo experimento. Nos 30 anos seguintes, técnicas de planejamento se propagaram entre os processos químicos e industriais. Tem havido uma considerável utilização de planejamento de experimentos em áreas, como, o setor de negócios, finanças e operações governamentais (Montgomery, 2009).

O planejamento experimental estatístico–em inglês, *Design of Experiments (DOE)*– é o processo de primeiro planejar um experimento para então coletar e analisar dados através de métodos estatísticos que possibilitam extrair conclusões válidas e objetivas. Existem dois aspectos em qualquer estudo experimental: o planejamento do experi-

mento e a análise estatística dos dados. Esses dois aspectos estão intimamente relacionados, pois o método de análise depende diretamente do planejamento empregado (Montgomery, 2009).

Em geral, experimentos são usados para estudar o desempenho de processos e sistemas. O processo ou sistema pode ser representado pelo modelo apresentado na Figura 4.1. O processo pode ser visualizado como a combinação de operações, máquinas, métodos, pessoas e outros recursos que transformam alguma entrada incorporando as variáveis de entrada x_i e z_j em uma saída, que tem uma ou mais respostas observáveis y. Algumas das variáveis de entrada são controláveis enquanto outras não.

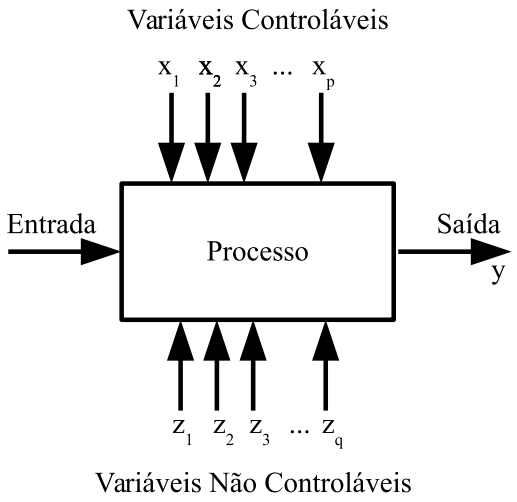


Figura 4.1: *Modelo genérico de um processo ou sistema (Montgomery, 2009).*

Um experimento é um teste, ou uma série de testes, no qual mudanças intencionais são aplicadas aos níveis das variáveis controláveis de um processo ou sistema, de forma que é possível examinar e identificar as causas das variações observadas na variável de resposta. Experimentos planejados são geralmente empregados sequencialmente. O primeiro experimento com um processo complexo, que tenha muitas variáveis controláveis, é quase sempre um experimento exploratório projetado para determinar quais dessas variáveis são mais importantes. Experimentos subsequentes são usados para refinar essa informação e determinar quais ajustes nos níveis são requeridos nessas variáveis críticas, de modo a melhorar o processo. E, se o objetivo do experimento é a otimização, determinar quais os níveis das variáveis críticas que resultam no melhor desempenho do processo (Montgomery, 2009).

4.2 Conceitos Gerais

A seguir apresentam-se alguns conceitos gerais e termos relacionados à aplicação de técnicas de planejamento e análise de experimentos.

- **Resposta.** São as variáveis dependentes que sofrem algum efeito nos testes quando estímulos são introduzidos propositalmente nas variáveis de entrada.

- **Fatores.** Fatores são variáveis de entrada que em princípio influenciam a resposta. Fatores controláveis são as variáveis de entrada controláveis que têm seus valores deliberadamente alterados no experimento para avaliar o efeito produzido na resposta. Os fatores não-controláveis são aqueles que, conhecidos ou não, influenciam na resposta, embora o experimentador não tenha como controlar os seus valores.

- **Níveis dos Fatores.** Os níveis são os valores atribuídos aos fatores no experimento. Esses valores podem ser quantitativos ou qualitativos. Em geral, o experimento é realizado para se descobrir quais são os níveis dos fatores que produzem a melhor resposta possível.

- **Tratamentos.** Um tratamento é o conjunto níveis específicos dos fatores de interesse. Assim, se os fatores de interesse são o tamanho da população do AE e a taxa de cruzamento, então um tratamento corresponde a níveis específicos do tamanho da população e da taxa de cruzamento usados em uma execução do AE.

- **Tentativa ou Corrida Experimental.** Uma tentativa ou corrida experimental corresponde a uma execução do processo estudado, que resulta em uma ou mais respostas observáveis.

- **Efeito Principal.** É a diferença média observada na resposta quando se muda o nível do fator investigado.

- **Efeito de Interação.** Uma interação ocorre quando um fator não produz o mesmo efeito na resposta nos diferentes níveis de outro fator, ou seja, um fator se comporta de forma distinta dependendo do nível do(s) outro(s) fator(es), em relação à resposta. A interação entre fatores é a medida de como o efeito de um fator depende dos níveis de um ou mais outros fatores. É a metade da

diferença entre os efeitos principais de um fator nos níveis de outro fator. Em geral, quando variáveis operam independentemente umas das outras, elas não exibem interação.

- **Matriz Experimental (ou Matriz do Planejamento).** É o plano formal construído para conduzir os experimentos.

Existem três princípios básicos que devem ser seguidos para a utilização de técnicas experimentais: *repetição*, *aleatorização* e *blocagem*.

A *repetição*, ou réplica, é o processo de repetir cada uma das combinações (linhas) da matriz experimental sob as mesmas condições de experimentação. Permite obter uma estimativa do erro experimental, uma unidade básica de medição para se determinar se as diferenças observadas nos dados são estatisticamente diferentes. Além disso, a repetição permite a obtenção de uma estimativa mais precisa dos efeitos.

A *aleatorização* é outro princípio experimental muito importante para evitar que desvios atípicos sejam obrigatoriamente associados a determinadas combinações de níveis. A ordem de realização das combinações de níveis deve ser aleatória. Desta forma, o efeito de fatores não-controláveis é minimizado.

O planejamento utilizando blocos – *blocagem* – tem como objetivo isolar o efeito de um ou mais fatores no resultado do experimento. Com isso, realiza-se o experimento em condições (blocos) mais homogêneas, e é possível evidenciar o efeito dos blocos.

4.3 Experimentos Fatoriais

Em um planejamento fatorial, o experimentador seleciona k fatores que podem causar um efeito na resposta do sistema. Então, os níveis – ou valores – para esses fatores são escolhidos. Se a escolha é por dois níveis, então o planejamento é conhecido como planejamento fatorial 2^k. Da mesma forma, um planejamento fatorial 3^k corresponde à escolha de 3 níveis.

Uma importante característica dos planejamentos fatoriais é que são a única maneira de se descobrir interações entre fatores. Quando vários fatores são de interesse em um experimento, um experimento fatorial deve ser usado.

A seguir são descritas variações do planejamento fatorial. Maiores detalhes sobre cada uma das técnicas descritas podem ser encontradas em (Montgomery, 2009).

4.3.1 Fatorial Completo

Os métodos de planejamento experimental são baseados em princípios estatísticos e são ferramentas importantes que auxiliam pesquisadores a extrair do sistema em estudo o máximo de informação útil, fazendo um mínimo de experimentos de forma racional e econômica. Um desses métodos é o método do planejamento fatorial completo, que possibilita a determinação da influência de uma ou mais variáveis sobre uma outra variável de interesse no resultado do experimento (Neto et al., 2010).

No planejamento fatorial completo, todas as combinações dos níveis são testadas. Por exemplo, um planejamento fatorial completo com cinco fatores e dois valores críticos requer $2^5 = 32$ corridas experimentais. Quando o algoritmo é um AE que inclui elementos aleatórios na sua estratégia de busca, réplicas de cada tratamento são necessárias para a coleta de dados significativos para análise (Adenso-Díaz e Laguna, 2006).

Os efeitos principais são causados pela influência de fatores principais, ou seja, a resposta sofre uma alteração quando o fator passa de seu nível inferior para o superior, ou vice-versa, independente da posição das demais. Os efeitos de interação são os que ocorrem quando a alteração da resposta com a mudança de nível de um fator depende do nível em que se encontra(m) outro(s) fator(es).

4.3.2 Fatorial com Blocos Completos Aleatorizados

A blocagem consiste em conter e avaliar a variabilidade produzida pelos fatores perturbadores – controláveis ou não-controláveis – do experimento. Essa técnica permite criar um experimento mais homogêneo e aumentar a precisão das respostas que são analisadas.

Na blocagem, sabe-se desde o início que certos fatores podem influenciar a resposta, porém não há interesse no efeito deles, e isso é levado em conta na definição do planejamento para evitar ou minimizar confundimentos. Por exemplo, variações nos resultados das execuções dos AE podem ser ocasionadas por diferenças na população inicial e pelo comportamento aleatório da implementação da mutação e cruzamento. Essas variações são diretamente dependentes da semente[1] utilizada na geração da sequência pseudo-

[1] A semente é um número inteiro utilizado pelo gerador de números pseudo-aleatórios (PRNG - *Pseudo Random Numbers Generators*, em inglês) para iniciar uma sequência de números. Sequências idênticas podem ser geradas repetidas vezes a partir de um mesmo valor de semente.

aleatória. Normalmente, a implementação de um AE não controla a semente, e seu comportamento é totalmente aleatório. Nesse caso, para demonstrar estatisticamente a significância de efeitos, é necessário um grande conjunto de dados. Outra forma de solucionar o problema do ruído causado pela semente é aplicar o planejamento com blocos completos aleatorizados (Czarn et al., 2004b).

No planejamento com blocos completos aleatorizados, toda combinação de níveis dos parâmetros aparece o mesmo número de vezes em cada bloco, e cada bloco usa a mesma semente. Réplicas de blocos idênticos, exceto pelo valor do fator bloqueado, são necessárias para verificar se os efeitos dos parâmetros são significativamente diferentes da variação devida às mudanças no fator bloqueado.

4.3.3 Fatorial com Dois Níveis 2^k

Um planejamento fatorial 2^k envolve k fatores, cada qual com dois níveis. Esses níveis podem ser quantitativos ou qualitativos. O nível de um fator quantitativo pode ser associado com pontos em uma escala numérica como, por exemplo, o tamanho da população ou o número de ilhas. Os níveis dos fatores qualitativos não podem ser dispostos pela sua ordem de magnitude, como os tipos de topologia de conexão entre ilhas e as estratégias de seleção.

Os dois níveis de um planejamento fatorial 2^k são referidos como nível baixo e nível alto, e denotados pelos sinais $-$ e $+$. Não importa qual valor é associado com o nível $+$ e o nível $-$, contanto que a associação permaneça consistente.

No início de um fatorial 2^k, fatores e níveis são especificados. Quando combinados os fatores e os níveis, tem-se a matriz do planejamento (ver, no Exemplo 4.1, a Tabela 4.2). Cada linha da matriz do planejamento corresponde a uma combinação de níveis, ou tratamento. Para cada tratamento, o processo em estudo é executado e a variável de reposta y é coletada.

Depois da coleta de dados, os efeitos dos fatores são calculados e, com os testes estatísticos apropriados, é possível determinar se a variação observada na resposta depende de modo estatisticamente significante dos valores de entrada. Existem vários softwares estatísticos que são úteis para se montar e analisar um fatorial 2^k. Neste trabalho, o software de código aberto R (R Core Team, 2012) foi utilizado nos cálculos estatísticos e geração de gráficos.

Planejamentos fatoriais de dois níveis são muito úteis em investigações preliminares,

quando se quer saber se determinados fatores têm ou não influência sobre sobre a resposta de um dado sistema. Em (Box et al., 2005), a importância dos planejamentos fatoriais com dois níveis é enfatizada através dos seguintes argumentos:

- Requerem relativamente poucas execuções de teste para cada fator.
- A interpretação das observações produzidas pelo planejamento pode ser realizada através da aplicação de aritmética elementar.
- O planejamento fatorial pode indicar tendências e a direção de próximos experimentos.
- O planejamento pode ser utilizado como blocos e aumentado quando é necessária uma exploração maior, um processo conhecido como montagem sequencial – em inglês, *sequential assembly*.
- São a base para o planejamento fatorial fracionário com dois níveis.
- O planejamento fatorial e o fatorial fracionário pertencem a uma estratégia sequencial de realização de experimentos.

4.4 Análise do Fatorial 2^k

Muitas implementações do fatorial 2^k contam com as réplicas para determinar a significância estatística dos efeitos estimados. Quando há réplicas, a determinação da relevância estatística dos efeitos é, em geral, obtida pela análise de variância ou pela regressão linear, quando os pressupostos de que os resíduos sejam variáveis aleatórias não correlacionadas, com média zero, variância constante e com distribuição próxima da normal são garantidos. Os resíduos são obtidos pela diferença entre uma observação real e o valor ajustado pelo modelo.

Box et al. (2005) tem uma regra informal: efeitos maiores que duas ou três vezes o erro padrão não são facilmente explicáveis apenas pelo acaso.

Os testes estatísticos formais são importantes para se apurar quais efeitos são devido ao erro amostral. A significância estatística não demonstra a significância prática, ou seja, se o efeito estimado é grande o suficiente para ser importante no experimento. Entretanto, um efeito não pode ser dito ter significância prática se não tem significância estatística (Rardin e Uzsoy, 2001).

Quando o fatorial 2^k não tem réplicas, existe o método de Daniel (1959), no qual efeitos são plotados em um gráfico de probabilidade normal. Os efeitos que são desprezíveis são normalmente distribuídos com média zero e variância σ^2 e, portanto, estarão dispostos ao longo de uma linha reta. Os efeitos significativos estarão dispostos distantes da linha reta, uma vez que possuem média diferente de zero. O modelo preliminar conterá os efeitos que são significantes segundo o gráfico de probabilidade normal, e uma estimativa do erro pode ser obtida com a combinação dos efeitos desprezíveis.

Caso um ou mais fatores não tenham efeito significativo, o planejamento fatorial 2^k pode ser reduzido, ou projetado em um fatorial com menos fatores. Em geral, se forem retirados j fatores, então permanecerão $r = k - j$ fatores e o planejamento original 2^k com n réplicas é projetado em um fatorial 2^r com $n2^j$ réplicas (Montgomery, 2009).

4.4.1 Modelo de Regressão do Fatorial 2^k

Os resultados do planejamento fatorial 2^k podem ser expressos em termos de um modelo de regressão linear. Para um fatorial 2^2, o modelo completo dos efeitos é dado pela equação:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \varepsilon, \tag{4.1}$$

onde y é a resposta (variável dependente), β's são os coeficientes cujos valores serão estimados, x_i são as variáveis preditoras (variáveis independentes), que representam os níveis codificados em -1 e $+1$, e ε é o termo do erro aleatório, cuja a distribuição é geralmente sujeita a restrições para possibilitar uma correta análise dos dados para modelos paramétricos, isto é, os erros serem variáveis normais $\varepsilon_{ij} \sim N(0, \sigma^2)$, com a mesma variância e independentes.

O método usado para estimar os coeficientes desconhecidos β's é o método dos mínimos quadrado (Kutner et al., 2005). Os coeficientes da regressão estão relacionados com os efeitos estimados. O coeficiente β_0 é estimado pela média de todas as respostas. Os valores estimados de β_1 e β_2 correspondem à metade do valor do efeito principal correspondente[2]. De mesma modo, os coeficientes de interação como, por exemplo, β_{12} são a metade do valor do efeito de interação dos fatores. Para maiores detalhes ver Montgomery (2009).

Na regressão, o coeficiente de determinação R^2 é uma medida estatística de quão

[2]Os coeficientes de regressão medem o efeito de uma variação unitária em x_i sobre a média de y e a estimativa do efeito está baseada na variação de duas unidades: de -1 a $+1$.

bem a linha da regressão se aproxima dos valores observados. O coeficiente de determinação ajustado $R^2 ajustado$ é quase o mesmo que o R^2, porém ele penaliza a estatística se mais variáveis são adicionadas ao modelo,

Outras medidas foram desenvolvidas para se avaliar a qualidade do modelo. O critério de informação *Akaike* (AIC), o teste *Chi-square*, o critério de validação cruzada, e outros (Busemeyer e Wang, 2000). Eles são úteis na etapa de seleção do modelo, uma das etapas da análise do fatorial 2^k, onde é verificado se todas as potenciais variáveis preditoras são necessárias ou apenas um subconjunto delas é adequado. O número de modelos possíveis cresce com o número de preditores, o que torna a seleção uma tarefa difícil. Existe uma variedade de funções disponíveis em softwares estatísticos que auxiliam nessa tarefa, como a função `stepAIC` do pacote `MASS` (Venables e Ripley, 2002) do R.

4.4.2 Verificação do Modelo

Para o modelo de regressão, algumas suposições devem ser satisfeitas: os erros devem ser independentes e possuir distribuição normal, com variância constante. Violações das suposições básicas e a adequação do modelo podem ser facilmente verificadas através do exame visual dos resíduos. Resíduos são a diferença entre os valores observados e os valores estimados pelo modelo. Se o modelo é adequado, o gráfico dos resíduos pelos valores preditos não deve apresentar nenhum padrão, isto é, os resíduos devem estar distribuídos de forma aleatória. Qualquer sugestão de padrão pode indicar algum problema, tal como um modelo inadequado devido a não linearidade, a omissão de variáveis preditoras importantes, a presença de valores discrepantes[3], entre outros.

Um procedimento para testar a suposição de normalidade consiste na construção de um gráfico de probabilidade normal dos resíduos. Se a distribuição dos erros é normal, os resíduos estarão posicionados ao longo de uma reta.

A variância constante pode ser verificada no gráfico dos resíduos pelos valores preditos. Nesse gráfico, os resíduos devem estar distribuídos de forma aleatória, sem mostrar qualquer padrão.

A verificação da independência pode ser feita com o gráfico dos resíduos pelo tempo, se for conhecido. Os resíduos devem flutuar de forma aleatória em torno da linha base zero.

[3]Pontos discrepantes – *outliers*, em inglês – são observações que são muito maiores ou muito menores do que as demais.

A análise visual dos resíduos é subjetiva, mas com frequência os gráficos dos resíduos podem revelar problemas de maneira mais clara do que os testes estatísticos formais. Testes formais como o teste de *Durbin-Watson*, o de *Breusch-Pagan* e o de *Shapiro-Wilk* podem verificar de maneira formal a presença de autocorrelação (dependência), a variância homogênea e a normalidade dos resíduos, respectivamente. O R possui pacotes com implementações desses testes. Tem a função `ncvTest` do pacote `car` (Fox e Weisberg, 2011) que implementa o teste de Breusch-Pagan para o teste de variância constante, a função `shapiro.teste` do pacote `stats` (R Core Team, 2012) para o teste de normalidade; e a função `dwtest` do pacote `lmtest` (Zeileis e Hothorn, 2002) para o teste de independência de Durbin-Watson.

Se ocorrer uma violação das suposições do modelo, existem duas alternativas: a) abandonar o modelo de regressão linear para desenvolver e usar um modelo mais apropriado; b) aplicar alguma transformação nos dados. A primeira alternativa pode resultar em um modelo mais complexo que o modelo da segunda alternativa. Às vezes, uma função não linear pode ser expressa como uma linha reta se uma transformação adequada for empregada. Mais informações sobre transformações pode ser encontrada em (Kutner et al., 2005; Montgomery, 2009).

A transformação dos dados na resposta é utilizada quando os resíduos indicam variâncias desiguais. As transformações mais utilizadas são $\sqrt{y}$, $ln\, y$ ou $1/y$. A transformação apropriada é escolhida pela forma mostrada pelo gráfico dos resíduos e valores preditos ou pela aplicação de métodos como o definido por Box e Cox (1964), no qual respostas y positivas são transformadas de acordo com a Equação 4.2.

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^{\lambda} - 1}{\lambda}, & \text{se } \lambda \neq 0 \\ \log(y_i), & \text{se } \lambda = 0. \end{cases} \tag{4.2}$$

O *software* estatístico R fornece função que calcula a verossimilhança perfilada do parâmetro λ. O valor de λ deve ser escolhido a fim de maximizar essa função.

4.4.3 Análise de Variância ANOVA

A análise de variância (ANOVA) é um teste estatístico usado para comparar as médias de duas ou mais amostras independentes e normalmente distribuídas. ANOVA produz a estatística F que calcula a razão da variância entre as médias e a variância

dentro das amostras (Kutner et al., 2005).

Os pressupostos da ANOVA são os mesmos do modelo de regressão expostos na seção anterior. A ANOVA é robusta à suposição de normalidade. Se a suposição de homogeneidade da variância é violado, a ANOVA é pouco afetada se o tamanho da amostra é igual nos tratamentos, ou seja, se o experimento é balanceado. Porém, se a suposição de independência não for atendida, a inferência da ANOVA pode ser seriamente afetada (Kutner et al., 2005).

Alguns autores, como Montgomery (2009), Box et al. (2005), Coffin e Saltzman (2000), argumentam que a ANOVA é robusta à violações da não homogeneidade das variâncias e da não normalidade dos resíduos, especialmente para tamanho de amostras grandes.

Segundo Spall (2010), quando os pressupostos da ANOVA não podem ser atendidos, é sempre possível calcular o erro padrão dos efeitos e aplicar o teste *t*, que é considerado robusto para desvios de normalidade da amostra quando o seu tamanho é grande. Para Box et al. (2005), uma medida prática é considerar que os efeitos, maiores que duas ou três vezes o seu erro padrão, não são facilmente explicáveis pelo acaso. Uma medida mais precisa pode ser fornecida pelo teste *t*, mas os autores recomendam que sejam mostrados apenas o efeito estimado e o seu erro padrão, deixando a escolha do nível de significância ao leitor.

Exemplo 4.1 [Planejamento fatorial 2^2 (Jain, 1991)] Um estudo do impacto do tamanho da memória (fator A) e da *cache* (fator B) no desempenho de uma *workstation* define dois níveis de cada um dos dois fatores. A resposta é medida em MIPS (milhões de instruções por segundo) e considera-se o nível de 5% de significância estatística. A Tabela 4.1 apresenta os níveis definidos para os fatores A e B.

Tabela 4.1: *Níveis dos fatores A e B.*

	Nível -	**Nível +**
Fator A	4 Mbytes	16 Mbytes
Fator B	1 Kbyte	2 Kbyte

A matriz do planejamento com as observações obtidas nas três réplicas dos ensaios e a média dos valores observados encontra-se na Tabela 4.2.

A partir das observações coletadas, é construída a matriz de planejamento para o cálculo dos efeitos, mostrada na Tabela 4.3. Os efeitos são calculados pela multiplicação da sua coluna pela coluna com a média das observações dividido por 2^{k-1}. A primeira coluna é chamada de coluna I ou coluna identidade. Ela corresponde à média geral do

Tabela 4.2: *Matriz do Planejamento e observações coletadas no planejamento fatorial 2^2 com três réplicas do Exemplo 4.1.*

	Fator A	**Fator B**	**Desempenho**			**Media**
i	**Memória**	**Cache**	y_{i1}	y_{i2}	y_{i3}	$\overline{y}$
1	4	1	15	18	12	15
2	16	1	45	48	51	48
3	4	2	25	28	19	24
4	16	2	75	75	81	77

experimento. A média é obtida ao se multiplicar a coluna I pela coluna $\overline{y}$ e dividir por 2^k, que neste caso é igual a quatro.

Tabela 4.3: *Cálculo dos efeitos do fatorial 2^2 com três réplicas do Exemplo 4.1.*

I	**A**	**B**	**AB**	y	$\overline{y}$	Resíduos
1	-1	-1	1	(15, 18, 12)	15	(0, 3, -3)
1	1	-1	-1	(45, 48, 51)	48	(-3, 0, 3)
1	-1	1	-1	(25, 28, 19)	24	(1, 4, -5)
1	1	1	1	(75, 75, 81)	77	(-2, -2, 4)
164	86	38	20		Total	
164/4	86/2	38/2	20/2			
41	**43**	**19**	**10**		**Efeitos**	

A análise de variância confirma a significância do efeito dos fatores, como é mostrado na Tabela 4.4. É confirmado que existe interação entre os fatores e, portanto, a interpretação dos seus efeitos deve ser feita em conjunto.

Tabela 4.4: *Análise de variância do Exemplo 4.1.*

Fonte de Variação	Graus de Liberdade	Soma dos Quadrados	Média Quadrática	F	Valor-p
A	1	5547	5547	435,059	2,93E-008
B	1	1083	1083	84,941	1,56E-005
AB	1	300	300	23,529	0,001271
Resíduos	8	102	12,7		

O modelo de regressão linear dos efeitos é dado por:

$$\hat{y} = 41 + 21,5x_1 + 9,5x_2 + 5x_1x_2 + \epsilon, \tag{4.3}$$

onde $\hat{y}$ é o desempenho predito, x_1 é fator tamanho da memória e x_2 o tamanho da *cache*, que representam os níveis codificados em -1 e $+1$, e ϵ é o termo do erro aleatório.

A Figura 4.2 mostra os gráficos dos resíduos do modelo de regressão linear dado pela Equação (4.3). Os gráficos mostram os resíduos distribuídos de forma aleatória e situados próximos da reta no gráfico de probabilidade normal. Nenhum indício de violação das suposições do modelo é visualizado.

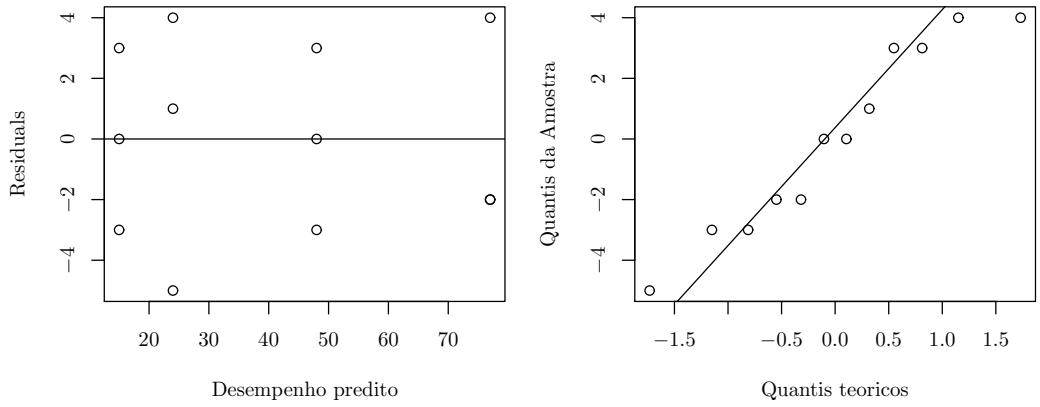
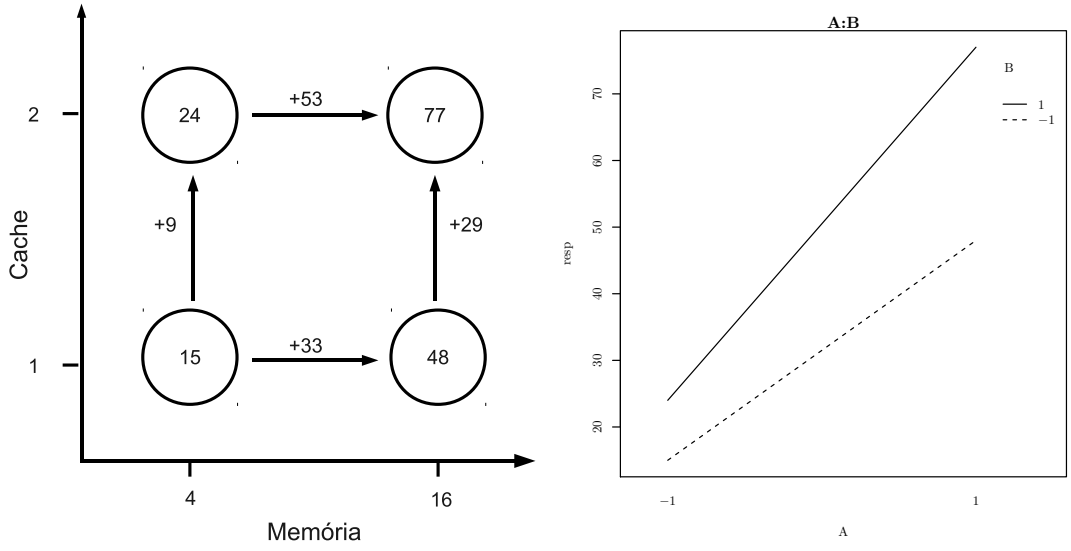


Figura 4.2: *Gráfico dos resíduos pelos valores preditos e gráfico de probabilidade normal dos resíduos do modelo de regressão do Exemplo 4.1.*



(a) Diagrama do fatorial 2^2. (b) Gráfico da interação A:B.

Figura 4.3: *Diagramas para interpretação dos resultados do planejamento fatorial 2^2 do Exemplo 4.1.*

Os gráficos da Figura 4.3 ajudam a interpretar os resultados do cálculo dos efeitos. O gráfico de interação exibido na Figura 4.3(b) mostra que o efeito de interação AB indica que uma mudança no nível da memória quando a cache está no seu nível + causa um maior aumento do desempenho do que quando a cache está no nível −.

O código R executado para analisar os dados e gerar os gráficos é apresentado na Figura 4.4.

□

4.5 Considerações do Capítulo

Neste Capítulo foram introduzidos os conceitos relacionados ao planejamento de experimentos. Maiores detalhes podem ser encontrados nos diversos livros publicados sobre o assunto, dentre eles temos (Montgomery, 2009), (Montgomery e Runger, 2009), (Wu e Hamada, 2000), (Neter et al., 1996), (Box et al., 2005).

```
> # leitura dos dados
> ws = read.table("jain2.txt", head=T)
> ws
    A  B resp
1  -1 -1   15
2   1 -1   45
3  -1  1   25
4   1  1   75
5  -1 -1   18
6   1 -1   48
7  -1  1   28
8   1  1   75
9  -1 -1   12
10  1 -1   51
11 -1  1   19
12  1  1   81
> # regressão linear
> ws.lm = lm(resp ~ (A+B)^2, data=ws)
> summary(ws.lm)
Call:
lm(formula = resp ~ (A + B)^2, data = ws)
Residuals:
   Min     1Q Median     3Q    Max
 -5.00  -2.25   0.00   3.00   4.00

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   41.000      1.031  39.776 1.76e-10 ***
A             21.500      1.031  20.858 2.93e-08 ***
B              9.500      1.031   9.216 1.56e-05 ***
A:B            5.000      1.031   4.851  0.00127 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.571 on 8 degrees of freedom
Multiple R-squared: 0.9855,        Adjusted R-squared: 0.9801
F-statistic: 181.2 on 3 and 8 DF,  p-value: 1.083e-07
> # exibir ANOVA
> anova(ws.lm)
Analysis of Variance Table
Response: resp
          Df Sum Sq Mean Sq F value    Pr(>F)
A          1   5547  5547.0 435.059 2.928e-08 ***
B          1   1083  1083.0  84.941 1.556e-05 ***
A:B        1    300   300.0  23.529  0.001271 **
Residuals  8    102    12.7
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> # plotar gráficos para verificar resíduos
> par(mfrow=c(1,2)); plot(ws.lm, which=c(1,2)); par(mfrow=c(1,1));
> # plotar gráfico de interação
> interaction.plot(ws$A,ws$B, ws$resp, ylab="resp", xlab="A", main="A:B", legend=T)
> ## Testes formais
> # teste normalidade
> shapiro.test(ws.lm$residuals)
        Shapiro-Wilk normality test
data:  ws.lm$residuals
W = 0.9289, p-value = 0.3684
> # teste homoscedastiscidade
> ncvTest(ws.lm)
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 0.03681301    Df = 1     p = 0.8478462
> # teste independência
> dwtest(ws.lm)
        Durbin-Watson test
data:  ws.lm
DW = 2.9608, p-value = 0.9683
alternative hypothesis: true autocorrelation is greater than 0
```

Figura 4.4: *Código R executado para análise do Exemplo 4.1.*

Capítulo 5

Estudo de Parâmetros de Metaheurísticas

O objetivo deste Capítulo é apresentar uma descrição geral dos principais enfoques para o estudo de parâmetros de metaheurísticas – em particular de AG, identificar as abordagens atuais, e dado o foco deste trabalho, um interesse especial é dado às abordagens que fazem uso de planejamento e análise de experimentos.

5.1 Introdução

Para ilustrar a complexidade do problema de configuração de parâmetros, os autores Adenso-Díaz e Laguna (2006) relatam uma evidência anedótica de que 10% do tempo total dedicado ao projeto e teste de uma nova metaheurística é gasto no seu desenvolvimento e os 90% restante é consumido pelo ajuste fino dos parâmetros. Supondo uma metaheurística simples com 2 parâmetros e 10 valores possíveis para cada um, há um total de 100 possibilidades de configurações. Além disso, o comportamento aleatório de algumas metaheurísticas, como os AG, faz com que seja necessário repetir as execuções, elevando ainda mais o número de testes.

Essa complexidade vai além do fato de haver um número muito grande de possibilidades de configuração de parâmetros a se escolher e pode ser vislumbrada através das seguintes observações:

- Apesar das metaheurísticas serem aplicadas extensivamente, ainda não há uma metodologia padrão para se explorar como e quais parâmetros afetam o desempenho. Com isso, a escolha dos valores dos parâmetros é realizada sem critério formal, baseada em experiências pessoais, escolhas *ad-hoc* ou em algumas poucas experimentações com diferentes configurações, em geral, variando um parâmetro

de cada vez (Birattari, 2004).

- Quando é realizada a experimentação para escolha da melhor configuração de parâmetros, nem sempre essa experimentação segue uma metodologia, e muitas vezes são adotados procedimentos não adequados, como descrito com maiores detalhes no Capitulo 6.

- A influência dos parâmetros no desempenho do algoritmo não pode ser tratada de forma independente. É preciso verificar se existe interação entre eles e qual o efeito dessa interação (Czarn et al., 2004b).

- O comportamento aleatório de algumas metaheurísticas dificulta a mensuração do efeito dos parâmetros no desempenho, pois são necessárias várias repetições da execução para garantir a sua significância (Alba, 2005; Czarn et al., 2004b). Isso implica em outra questão em aberto: quantas repetições são suficientes para garantir resultados significativos sem desperdício de recursos. Em outras palavras: qual o tamanho n da amostra necessário para se detectar um dado efeito com potência conhecida e dado um certo nível de significância α. A significância de um resultado dever ser comprovada por ferramentas estatísticas, porém nem tudo que é estatisticamente significativo possui relevância prática (Bartz-Beielstein et al., 2010; Rardin e Uzsoy, 2001).

- O teorema da inexistência de almoço grátis – em inglês, *no free lunch theorem (NFLT)* – afirma que nenhuma metaheurística é melhor que outra quando a comparação é realizada sobre o conjunto de todos os problemas matematicamente possíveis (Wolpert e Macready, 1997, 1996). Da mesma forma, a configuração ótima dos parâmetros de um algoritmo depende da instância do problema a ser resolvido. Em (Czarn et al., 2004b), são apresentadas diferentes configurações ótimas de parâmetros para o mesmo AG aplicado a diferentes funções. Segundo Eiben e Jelasity (2002), existe a necessidade de se classificar os problemas de modo que fique mais fácil generalizar resultados encontrados para outros problemas similares.

Na seção 5.2 são introduzidos os parâmetros dos AE – classe a qual os AG pertencem – seus domínios e a definição de alguns termos que serão utilizados nas seções seguintes.

5.2 Configuração de Parâmetros

A aplicação de uma metaheurística na solução de problemas resume-se, em linhas gerais, na configuração de seus parâmetros para sua melhor adequação à instância do problema a ser resolvido. A configuração de parâmetros é fundamental para seu bom desempenho em relação à qualidade da solução e ao tempo de processamento gasto.

Um dos primeiros estudo sobre a configuração de parâmetros dos AE mostra que a determinação dos parâmetros pode ser classificada em: *ajuste de parâmetros* e *controle de parâmetros* (Eiben et al., 1999). No caso do controle de parâmetros, os valores iniciais atribuídos aos parâmetros são modificados durante a execução do algoritmo por alguma estratégia de controle – pode ser uma estratégia determinística, adaptativa ou autoadaptativa. O ajuste de parâmetros define os valores dos parâmetros antes da execução, e esses não mudam durante a execução. A abordagem deste trabalho pode ser classificada como ajuste de parâmetros, pois os valores são definidos antes da execução do algoritmo.

O problema de ajuste pode ser caracterizado como um problema de busca pela configuração ótima. Segundo Eiben e Smith (2011), o problema de ajuste de parâmetros corresponde a uma busca no espaço de vetores de parâmetros onde uma função utilidade retorna o valor da medida de desempenho do algoritmo para um dado vetor de parâmetros. Soluções do problema de ajuste são vetores de parâmetros com utilidade máxima. Desta forma, o problema de ajuste de parâmetros depende dos seguintes itens: (1) os problemas a serem resolvidos, (2) o algoritmo utilizado, (3) a função utilidade, e (4) o método de ajuste. Assim como a função *aptidão* retorna a qualidade da solução candidata nos AE, a função *utilidade* retorna a qualidade do vetor de parâmetros. Uma diferença importante entre as duas funções é que a função aptidão é, em geral, determinística – dependendo da instância do problema a ser resolvido – e a função utilidade é sempre estocástica, pois corresponde ao desempenho de um algoritmo que é estocástico. Isso dificulta a comparação entre os valores da função utilidade, uma vez que os dados apresentam uma grande variância.

A Tabela 5.1 mostra o vocabulário que distingue as principais entidades no contexto da resolução de problemas e do ajuste de parâmetros.

Os vários parâmetros de uma metaheurística possuem diferentes domínios. Por exemplo, o domínio do parâmetro *tipo de cruzamento* de um AE, tipicamente, é o

Tabela 5.1: *Vocabulário que distingue as principais entidades no contexto da resolução de problemas e ajuste de parâmetros.*

	Resolução de Problemas	Ajuste de Parâmetros
Método	Algoritmo evolutivo	Método de ajuste
Espaço de Busca	Vetores de soluções	Vetores de parâmetros
Qualidade	Aptidão	Utilidade
Avaliação	Estimação	Teste

conjunto de valores formado pelas opções de cruzamento: um ponto, dois pontos, uniforme ou médio. Esse domínio corresponde a um conjunto de valores que não possui métrica razoável de distância ou de ordem entre os seus elementos. Já o domínio do parâmetro *taxa de cruzamento* é um subconjunto dos números reais $\mathbb{R}$ e possui uma métrica de distância e uma ordem entre os seus elementos.

O domínio dos parâmetros pode ser classificado em *qualitativo* ou *quantitativo*. Parâmetros quantitativos são em geral valores numéricos como, por exemplo, os valores da *taxa de cruzamento* que pertencem ao intervalo $[0, 1]$. Os parâmetros qualitativos são simbólicos como, por exemplo, os valores do tipo de cruzamento que pertencem ao conjunto [*um ponto, dois pontos, uniforme, médio*]. Esses valores não possuem uma ordem ou pré ordem, nem uma estrutura de busca.

Os autores Eiben e Smith (2011) ainda distinguem o ajuste de parâmetros em ajuste *estrutural* e o ajuste *paramétrico*. O primeiro está relacionado ao ajuste dos parâmetros qualitativos e o segundo aos parâmetros quantitativos.

5.3 Objetivo do Ajuste de Parâmetros

O ajuste de parâmetros tem como objetivo geral melhorar a qualidade do algoritmo. A qualidade do algoritmo é determinada por duas características principais: desempenho e robustez.

Em geral, o desempenho é medido pela qualidade da solução obtida e pela rapidez do algoritmo. Na seção 6.5 são apresentadas várias opções de mensuração do desempenho. Em (Eiben e Smith, 2011), os autores deixam claro que a medida de desempenho determina a escolha da melhor configuração de parâmetros, ou seja, qualquer declaração sobre os melhores valores dos parâmetros deve vir acompanhada do relato de qual foi a medida de desempenho utilizada.

A robustez está relacionada com a variação do desempenho do algoritmo através

de algumas dimensões. Essas dimensões é que possibilitam diferentes visões sobre a robustez. Para uma instância de um AE, a variação do desempenho pode ser considerada em relação a: (1) os valores dos parâmetros, (2) as instâncias de problemas, e (3) a semente do gerador pseudo-aleatório. No primeiro caso, a robustez relativa a mudanças de valores de parâmetros caracteriza um AE como *tolerante*. No segundo caso, busca-se o melhor ajuste de parâmetros que obtenha um desempenho razoável para um grande número de problemas. Nesse caso, o AE é considerado *amplamente aplicável*. Por último, a robustez relativa a diferentes sementes pode ser medida pela taxa de sucesso – porcentagem de execuções que ao terminar fornecem a solução esperada – e, caso a taxa de sucesso seja alta, o AE é considerado *bem sucedido*.

5.4 Abordagens para o Ajuste de Parâmetros

De acordo com Eiben e Smith (2011), existem três maneiras diferentes de se distinguir os métodos de ajuste de parâmetros de AE baseando-se nas diferentes noções de robustez e na preocupação em se obter maior conhecimento sobre o algoritmo ajustado. As duas primeiras taxonomias descritas a seguir são fortemente orientadas por diferenças entre o algoritmo interno de cada ajustador.

- **Taxonomia T_1.**

  De modo geral, todos os algoritmos de ajuste seguem o princípio do "gerar–e–testar", isto é, gerar valores para os parâmetros e testar o desempenho do algoritmo. Esses algoritmos podem ser classificados em: (1) ajustadores *não-iterativos* ou (2) ajustadores *iterativos*. Os não-iterativos realizam o passo "gerar" somente uma vez na inicialização, criando um conjunto fixo de vetores de parâmetros a serem avaliados durante a etapa "testar". Assim, são encontrados os melhores valores dentre os elementos do dado conjunto inicial.

  A segunda categoria de algoritmos ajustadores é formada por métodos iterativos que iniciam com um conjunto inicial pequeno, e novos vetores são criados iterativamente durante a execução. Exemplos desse tipo de método são os *meta-Algoritmos Evolutivos* e *Métodos Iterativos de Amostragem*.

  Dada a natureza estocástica dos AE, vários testes são necessários para a estimação confiável da utilidade. Desta forma, existem os procedimentos de *único-estágio* e os de *múltiplos-estágios*. Os primeiros executam o mesmo número de testes para

um dado vetor, enquanto os últimos usam estratégia mais sofisticada, em geral, aumentando o passo "testar" adicionando um passo "selecionar", onde os vetores mais promissores são selecionados para mais testes, e os menos promissores são ignorados.

- **Taxonomia T_2.**

  A taxonomia T_2 é baseada no esforço computacional. O esforço dispendido por um ajustador é a combinação de três custos: (1) número de vetores de parâmetros a serem testados pelo ajustador; (2) número de testes – execuções do AE – por vetor de parâmetros para estabelecer sua utilidade; e (3) número de avaliações da função em uma execução do AE.

  No primeiro, os métodos iniciam com um conjunto pequeno de vetores de parâmetros e iterativamente geram novos conjuntos de vetores de modo inteligente, isto é, os novos vetores de parâmetros são provavelmente bons em termos de sua utilidade.

  O segundo traz o problema da determinação do número de repetições de testes por vetor de parâmetros, que está relacionado com o cálculo da potência do teste estatístico. Um número baixo pode impossibilitar a confirmação estatística das diferenças entre os valores da utilidade de dois vetores de parâmetros. Muitos testes podem melhorar a confiança nas comparações entre os vetores, mas trazem um custo adicional. Os métodos desse grupo executam poucos testes por vetor no início e aumentam esse número até o valor mínimo necessário para garantir que as comparações sejam estatisticamente significantes. São conhecidos como métodos estatísticos exploratórios – em inglês, *screening* –, de classificação e de seleção. São aplicáveis a parâmetros quantitativos e qualitativos.

  Em relação ao terceiro e último custo, Eiben e Smith (2011) dizem não conhecer um método que economiza no número de avaliações de função por execução.

- **Taxonomia T_3.**

  A taxonomia T_3 é baseada nos objetivos do ajuste: encontrar o melhor vetor de parâmetros de melhor desempenho e obter o máximo de conhecimento sobre a robustez do AE. O primeiro pode ser relacionado com a busca em profundidade – *exploitation* –, e o segundo pode ser relacionado com a busca em largura – *exploration*. Muitos dos métodos de ajuste são capazes de retornar, além dos melhores

valores para os parâmetros, informações sobre os indicadores de robustez.

De acordo com a distinção entre a capacidade de cada método de realizar a busca em profundidade e/ou em largura, são identificados quatro categorias: (1) métodos por amostragem – largura; (2) métodos baseados em modelo – ambos; e (3) métodos exploratórios – ambos; (4) meta-algoritmos evolutivos – profundidade.

– **Métodos por Amostragem.** São métodos que procuram reduzir o número de vetores de parâmetros a serem testados com planejamento experimental.

  Os dados obtidos devem ser analisados em etapa posterior para definir os melhores valores dos parâmetros e quais são os mais robustos. Em geral correspondem à fase inicial dos métodos baseados em modelos. A falta de refinamento na busca leva a vetores de parâmetros de baixa qualidade e proveem informação limitada sobre a robustez.

  Os métodos iterativos de amostragem refinam os pontos de amostragem a cada iteração. O método CALIBRA (Adenso-Díaz e Laguna, 2006) é um exemplo de método de amostragem iterativo. Ele inicia com um planejamento fatorial completo com base no primeiro e terceiro quartil dentro do intervalo de cada parâmetro. A partir desses resultados, novos vetores são gerados para a próxima iteração através de um arranjo ortogonal de *Taguchi* com três níveis (Roy, 2010). Esse procedimento é repetido até que o número máximo de testes seja alcançado.

  Métodos de amostragem iterativa necessitam de uma medida de qualidade para definir áreas de pontos de amostragem promissores, e somente podem ser usados para otimização do desempenho de AE, não para a robustez. Entretanto, a minuciosa varredura de parâmetros realizada permite uma análise detalhada da robustez no espaço de parâmetros e produz soluções de qualidade razoável.

– **Métodos Baseados em Modelos.** O objetivo desse método é minimizar o número de testes através da construção de um modelo da utilidade. O modelo é uma fórmula que mapeia os valores dos parâmetros a uma estimativa do valor da utilidade $\hat{u}$, como mostra a Equação 5.1, para um exemplo

com dois parâmetros quantitativos p_1 e p_2.

$$\hat{u}(p_1, p_2) = \beta_0 + \beta_1 \cdot p_1 + \beta_2 \cdot p_1^2 + \beta_3 \cdot p_2 + \beta_4 \cdot p_2^2 + \beta_5 \cdot p_1 \cdot p_2. \quad (5.1)$$

Onde os valores β correspondem ao peso associado ao parâmetro. Um valor alto de β indica uma grande influência na utilidade quando o parâmetro é variado. Essa fórmula permite estimar a tolerância do algoritmo para um dado intervalo de desempenho. Se o método baseado em modelo é aplicado para múltiplos problemas, então a diferença nos valores de β também indicam uma baixa robustez a mudanças nos problemas. Por serem métodos de amostragem de um único estágio para geração de uma população, a qualidade do melhor vetor encontrado e a qualidade do modelo é relativamente baixa. Esses pontos negativos são superados com os métodos iterativos descritos a seguir.

Métodos iterativos baseados em modelos são uma extensão dos métodos baseados em modelos, com o acréscimo de um passo onde um procedimento de busca local é aplicado para otimizar os valores dos parâmetros.

Um exemplo desse tipo de método é apresentado no trabalho de Coy et al. (2001). O primeiro estágio consiste em um planejamento fatorial completo que resulta em dados para construir um modelo de regressão linear e para determinar o caminho da descida mais íngreme. No segundo estágio, o caminho é percorrido e novos vetores são gerados e testados até que a melhor solução encontrada não melhore depois de um número específico de passos. A qualidade do melhor vetor de parâmetros encontrado depende fortemente da adequação do modelo construído no primeiro estágio.

Outro exemplo, é o método *Sequential Parameter Optimization* (SPO) (Bartz-Beielstein, 2010) que executa um procedimento com múltiplos estágios onde o modelo é constantemente atualizado. Cada iteração começa com a geração de um conjunto de novos vetores e a previsão de suas utilidades são então testadas para confirmar o seu valor de utilidade. Esses valores de utilidade são usados para atualizar o modelo na próxima iteração. Quando o número máximo de testes é alcançado, o procedimento termina. A precisão do modelo e a qualidade do melhor vetor de parâmetros encontrado dependem do

tipo de modelo utilizado. Embora possa ser qualquer modelo, como árvores de regressão ou lógica de regressão, é utilizado o método de *Kriging* (Isaaks e Srivastava, 1989) para modelar a utilidade. Os resultados obtidos pelo SPO foram melhores em termos de qualidade dos vetores de parâmetros encontrados quando comparados com os encontrados por meta-algoritmos evolutivos (Smit e Eiben, 2009).

– **Métodos Exploratórios.** A ideia destes métodos é identificar o melhor vetor de parâmetros a partir de um conjunto de vetores com número mínimo de testes. De forma iterativa, os vetores mais promissores são escolhido para serem re-testados numa próxima etapa. Assim é possível encontrar o melhor vetor de parâmetros com menos esforço computacional que os métodos por amostragem, ou podem investigar um conjunto maior de vetores de parâmetros com o mesmo esforço computacional.

  Existem os métodos exploratórios iterativos, como o método F-RACE (Birattari, 2004) que foi melhorado por Balaprakash et al. (2007) com o objetivo específico de combinar métodos exploratórios e busca refinada. O método inicia com uma região do tamanho do espaço de parâmetros, e essa região é usada para tirar amostras de uma população relativamente pequena de vetores. Usando a técnica do F-RACE, o número de vetores dessa população é reduzido até uma certa condição ser encontrada. Entretanto, ao contrário do F-RACE original, este é apenas o início do procedimento. Uma distribuição normal multivariada é ajustada aos vetores sobreviventes. Essa distribuição é usada como a função densidade de probabilidade para amostrar pontos para a nova população. O procedimento completo de exploração e geração de novos pontos pode ser repetido até o número máximo de testes ser alcançado. Esse método pode ser visto como uma forma básica de um método iterativo baseado em modelo, ou uma forma especial de meta-algoritmo evolutivo e, portanto, pode fornecer bom desempenho e informações importantes sobre a robustez.

– **Meta-Algoritmos Evolutivos.** O problema de encontrar vetores de parâmetros com alta utilidade é uma tarefa de otimização complexa com uma função objetivo não linear, variáveis que interagem, vários ótimos locais, ruído e falta de resolvedores analíticos. A princípio, este é exatamente o

tipo de problema onde os AE são heurísticas competitivas. É natural, então, a ideia de usar uma abordagem evolutiva para otimizar os parâmetros de um AE.

A ideia de um meta-Algoritmo Evolutivo – meta-AE – foi introduzida em 1978 (Mercer e Sampson, 1978) *apud* (Eiben e Smith, 2011), mas devido ao grande custo computacional a pesquisa foi muito limitada. Na década de 1980, J. Greffenstette (Grefenstette, 1986) conseguiu mostrar que esta abordagem era efetiva. Basicamente qualquer AE pode ser usado como um meta-AE onde os indivíduos são os vetores de parâmetros e a aptidão é dada pela função utilidade. Apesar dos meta-AE serem bons para encontrar vetores de qualidade, eles não fornecem nenhum modelo do problema, nem maiores informações sobre a robustez do algoritmo.

Um meta-AE melhorado – *Relevance Estimation e Value Calibration* (REVAC) – foi proposto por Nannen e Eiben (2007). A população aproxima a função densidade de probabilidade das áreas mais promissoras do espaço de vetores de parâmetros. Essa função pode ser usada para analisar a sensibilidade e a relevância dos parâmetros, e o custo de se ajustar cada parâmetro. Porém, não identifica interações entre os parâmetros.

Algumas vezes, a aplicação ou os requisitos dos usuários demandam que múltiplos problemas ou indicadores de desempenho sejam considerados.

5.5 Estatística no Estudo de Parâmetros

O uso de procedimentos estatísticos no estudo de parâmetros significa implementar métodos estatísticos computacionalmente exaustivos, que permitam obter um conjunto de valores de parâmetros adequados através de provas rigorosas. Os métodos estatísticos são submetidos a uma série de passos planejados para permitir coletar dados, de tal forma que esses possam ser analisados por métodos estatísticos que resultem em conclusões válidas e objetivas.

Existem diversos métodos estatísticos que podem ser utilizados no ajuste de parâmetros, tais como: o planejamento fatorial, análise de variância, análise de regressão, entre outros. A seguir, são descritos alguns dos métodos utilizados em estatística para calibrar os parâmetros em algoritmos, assim como trabalhos recentes que fazem uso

desses métodos.

Um experimento é um teste ou uma sequência de testes de um processo, onde as variáveis de entrada são modificadas intencionalmente com a finalidade de estudar seus comportamentos e seus efeitos na saída desse processo. O planejamento de experimentos, como exposto no Capítulo 4, é o procedimento de primeiro planejar um experimento para então coletar e analisar esses dados através de métodos estatísticos e obter conclusões válidas e objetivas. Existem dois aspectos em qualquer estudo experimental: o planejamento do experimento e a análise estatística dos dados. Os dois estão intimamente relacionados pois o método de análise depende diretamente do planejamento empregado (Montgomery, 2009). No Capítulo 4 podem ser encontradas informações mais detalhadas sobre as técnicas de planejamento experimental.

No estudo de Coy et al. (2001), os autores propõem um procedimento para ajustar os parâmetros de uma heurística de busca local baseada no método de relaxação Lagrangiana aplicada na resolução de problemas de roteamento de veículos – em inglês, *vehicle routing problem (VRP)*. A Tabela 5.2 apresenta um resumo do trabalho. Os casos de teste apresentados mostram o ajuste de seis parâmetros. Os passos do procedimento proposto são mostrados a seguir:

- Passo 1 – Selecionar as instâncias do problema.
  Escolher as instâncias do problema VRP de forma que o subconjunto contenha amostras da maioria das diferenças estruturais, por exemplo, distribuição de demanda e distribuição de clientes encontrados no conjunto de problemas. Selecionar o mínimo de problemas possíveis.

- Passo 2 – Selecionar o nível inicial de cada parâmetro, o intervalo que os valores de cada parâmetro pode variar e a quantidade que cada parâmetro muda. É o *estudo piloto*. Nos estudos de caso apresentados, foi utilizada a abordagem da tentativa–e–erro para determinar os intervalos de valores dos parâmetros.

- Passo 3 – Selecionar conjuntos de parâmetros bons para cada instância do problema do conjunto de análises usando planejamento de experimentos. Neste passo, é aplicado o planejamento fatorial fracionário $2^{6-1}-$ com cinco réplicas, cada uma partindo de uma solução inicial gerada aleatoriamente. Com os dados obtidos, é realizada a regressão linear que resulta numa superfície de resposta. Através do gradiente descendente, é definido o caminho mais íngreme. Para cada

passo do caminho, a heurística é executada cinco vezes com a mesma solução inicial, até o limite da região experimental ou até que o desempenho da heurística não apresente melhora.

- Passo 4 – Combinar os valores obtidos no Passo 3 para cada uma das instâncias do problema definidas no Passo 1.

O método de Coy et al. (2001) não prevê o caso em que um parâmetro possui valor qualitativo. Isso restringe a aplicação da superfície de resposta uma vez que parâmetros qualitativos não possuem uma ordem ou métrica de distância entre os elementos do seu domínio.

Tabela 5.2: *Resumo da análise experimental apresentada por Coy et al. (2001).*

Resumo	
Objetivo:	Ajustar parâmetros.
Fatores:	Seis fatores.
Medida de desempenho:	Solução média.
Blocagem:	—
Planejamento:	Planejamento Fatorial Fracionário 2^{6-1}.
Réplicas:	Cinco réplicas.
Critério de parada:	Solução ótima.
Algoritmo:	Busca local relaxação Lagrangiana.
Parâmetros fixos:	—
Problemas:	34 instâncias do problema de roteamento de veículos.
Análise:	Teste de significância, regressão linear, superfície de resposta.
Observações:	Apenas valores quantitativos; algoritmos determinísticos;

Outro ponto a ressaltar é que as heurísticas, utilizadas nos estudos de caso apresentados, são duas versões da heurística de busca local baseada no método exato da relaxação Lagrangiana, ou seja, algoritmos determinísticos. A aleatoriedade está presente na solução inicial utilizada pela heurística e as execuções são repetidas cinco vezes para, segundo os autores, amenizar a variação nos dados causada por essa aleatoriedade. Esse número de repetições é relativamente pequeno quando comparado com o número de repetições utilizados em experimentos com metaheurísticas estocásticas como os AG. O método proposto por Czarn et al. (2004b), mostrado mais adiante, faz uso de 100 a 500 repetições da execução de um AG.

Czarn et al. (2004b) apresentam uma metodologia estatística para o estudo exploratório dos AG resumida na Tabela 5.3. Através da análise de variância (ANOVA), examinam a relação entre os operadores de cruzamento e mutação para quatro funções de teste. Aplicando o Planejamento com Blocos Completos Aleatorizados para

Tabela 5.3: *Resumo da análise experimental apresentada por Czarn et al. (2004b).*

Resumo	
Objetivo:	Avaliar a influência dos operadores de cruzamento e mutação sobre o desempenho dos AG.
Fatores:	Taxa de cruzamento e taxa de mutação.
Medida de desempenho:	Quantidade de gerações.
Blocagem:	Semente do gerador pseudo-aleatório.
Planejamento:	Planejamento de Blocos Completos Aleatorizados.
Réplicas:	Inicial de 100 réplicas e aumento até atingir a potência acima de 80%.
Critério de parada:	Solução ótima.
Algoritmo:	AG.
Parâmetros fixos:	Cromossomo binário de 22 bits, população de 50 indivíduos, seleção probabilística, cruzamento de um ponto, mutação de bit aleatória.
Problemas:	F_1 *Sphere*, F_3 *Step*, F_2 *Rosenbrock* e F_6 *Schaffer's*.
Análise:	ANOVA, superfície de resposta.
Observações:	Não apresenta tempos de execução; estuda apenas dois fatores.

controlar a variação causada pela semente do gerador de números aleatórios. O método apresentado pelos autores consiste nos passos mostrados a seguir, os quais são executados para cada uma das funções de teste:

- Passo 1 – Identificar fontes de variação e gerar blocos de execução isolando a fonte de ruído.

  A variação nos resultados das execuções nos AG deve-se às diferenças na população inicial e no comportamento aleatório da implementação da mutação e cruzamento. Essa variação é diretamente dependente da semente utilizada na geração da sequência pseudo-aleatória. Normalmente, a implementação de um AG não controla a semente e seu comportamento é totalmente aleatório. Nesse caso, para demonstrar estatisticamente a significância de efeitos é necessário um grande conjunto de dados. Outra forma de solucionar o problema do ruído causado pela semente é aplicar o *Planejamento com Blocos Completos Aleatorizados*. Nesse planejamento, toda combinação de níveis dos parâmetros aparece o mesmo número de vezes no mesmo bloco e cada bloco usa a mesma semente. Sementes são blocadas para garantir que as sementes usadas para implementar a inicialização da população, seleção, cruzamento e mutação sejam idênticas em cada bloco. Réplicas de blocos idênticos, exceto pela semente, são necessárias para verificar se os efeitos dos parâmetros são significativamente diferentes da variação devida

à mudanças na semente.

Os autores ressaltam que pouca atenção tem sido dada à blocagem da semente como fonte de variação ou ruído. Diferentes sementes do gerador de números pseudo-aleatórios na inicialização da população e na implementação da seleção, cruzamento e mutação, geram resultados diferentes para a mesma configuração de parâmetros sobre o mesmo problema. Para limitar essa fonte de ruído, é preciso blocar a semente através do agrupamento de unidades experimentais, para que cada execução do AG com níveis diferentes de taxa de cruzamento e mutação ocorram com as mesmas sementes.

- Passo 2 – Usar procedimentos para minimizar a ocorrência de observações censuradas[1] e para definir os intervalos de valores dos parâmetros.

- Passo 3 – Gerar um conjunto de dados iniciais com número arbitrário de réplicas. Em geral, os autores constataram que 100 réplicas foi um bom número inicial.

- Passo 4 – Calcular a potência do teste *post hoc* baseada em um tamanho de efeito escolhido.

  A potência do teste é a probabilidade $(1 - \beta)$ de rejeitar a hipótese nula quando ela é falsa. Uma potência igual ou maior que 80% ($\beta \leq 0.2$) é o que convencionalmente se deseja. O valor de β está relacionado ao tamanho da amostra. Uma amostra de tamanho muito pequeno em geral não produz resultados significativos, enquanto que um tamanho de amostra muito grande pode dificultar a análise e desperdiçar recursos.

  Os autores fazem uso dos índices d e f, índices do tamanho do efeito sem unidade, ambos definidos por Cohen (1988), no cálculo da potência. O tamanho da amostra é continuamente aumentado em cinco unidades até que seja obtida a potência maior ou igual a 80% no teste de detecção de uma interação entre cruzamento e mutação.

  O cálculo da potência do teste e do tamanho da amostra para determinada potência têm sido ignorados em vários estudos. Isso significa que não é certeza que os estudos realizados foram executados com a potência do teste adequada e, por

[1]Uma observação é censurada se não pode ser medida de forma precisa, e sabe-se que está além, ou aquém, de um limite. Por exemplo, quando o AG atinge um limite de esforço computacional sem suceder em obter a solução para o problema.

consequência, o tamanho da amostra adequada para detectar diferenças de interesse do experimento. Tamanho da amostra muito pequeno, em geral, resulta em efeitos de interesse serem considerados sem significância estatística. O problema se agrava quando não é feita a blocagem, e efeitos podem não ser detectados por causa do ruído produzido pela semente nos dados. Nesse caso, um tamanho muito maior de amostra é necessário para detectar efeitos de interesse.

- Passo 5 – Conduzir a análise ANOVA conjunta — em inglês, *pooled ANOVA* — e determinar quais parâmetros são estatisticamente significantes.
  Para comparar o desempenho de dois ou mais parâmetros usando o planejamento com blocos completos aleatorizados, é aplicada a análise de variância – ANOVA. A hipótese nula é que as médias para diferentes níveis dos parâmetros são iguais. A hipótese alternativa é que as médias não são iguais, e conclui-se que o parâmetro tem um efeito sobre a variável de resposta. Na ANOVA, os *p-valores* são válidos se as respostas são normalmente distribuídas. Entretanto, desvios moderados da normalidade não necessariamente implicam em séria violação nos pressupostos da ANOVA (Montgomery, 2009), especialmente para tamanho de amostras grandes. É procedimento padrão usar métodos como plotar um histograma com os resíduos ou construir um gráfico de probabilidade normal para verificar a normalidade da amostra da população. ANOVA permite o teste da significância de parâmetros individuais, ou seja, demonstrar estatisticamente o efeito do cruzamento e mutação, e da interação entre esses parâmetros.

- Passo 6 – Usar a regressão polinomial para obter coeficientes para a superfície de resposta, no caso da interação entre parâmetros ser estatisticamente significante, ou para obter coeficientes para uma superfície de resposta para cada parâmetro separadamente, caso a interação entre parâmetros não seja estatisticamente significante.

- Passo 7 – Diferenciar e resolver a superfície de resposta para cada parâmetro, obter os melhores valores e calcular os intervalos de confiança.

O trabalho apresentado por Czarn et al. (2004b) é um bom exemplo de uma análise experimental bem realizada e de acordo com as diretrizes descritas no Capítulo 6. Entretanto, os tempos de processamento não são relatados como é sugerido por Johnson

(2002) e, portanto, não é possível dimensionar o tempo gasto no experimento. Foram analisados apenas dois fatores quantitativos, a taxa de cruzamento e a taxa de mutação, e outros fatores que influenciam o desempenho dos AG não foram estudados.

Em outro trabalho dos mesmos autores Czarn et al. (2004a) (Tabela 5.4) busca-se respostas sobre duas questões: quais tipos de funções são suscetíveis a mostrar interação entre cruzamento e mutação e qual é a implicação prática da interação na obtenção de valores ótimos para esses parâmetros. Para a primeira questão, os autores constatam que à medida que a função de teste aumenta em modalidade, a interação entre cruzamento e mutação torna-se estatisticamente significante. Os autores supõem que, para funções altamente modais, a possibilidade de interação entre cruzamento e mutação deve ser considerada. Em relação à segunda questão, a implicação prática da interação é que quando tenta-se fazer o ajuste fino de um AG na resolução de um problema altamente modal, as taxas ótimas de cruzamento e mutação não podem ser obtidas de forma independente.

Tabela 5.4: *Resumo da análise experimental apresentada por Czarn et al. (2004a).*

Resumo	
Objetivo:	Avaliar a relação entre a ocorrência de interação entre cruzamento e mutação estatisticamente significante, e o aumento da modalidade de um problema.
Fatores:	Taxa de cruzamento e taxa de mutação.
Medida de desempenho:	Quantidade de gerações.
Blocagem:	Semente do gerador pseudo aleatório.
Planejamento:	Planejamento de Blocos Completos Aleatorizados.
Réplicas:	Inicial de 100 réplicas e aumento até atingir a potência acima de 80%.
Critério de parada:	Solução ótima.
Algoritmo:	AG.
Parâmetros fixos:	Cromossomo binário de 22 bits, população de 50 indivíduos, seleção probabilística, cruzamento de um ponto, mutação de bit aleatória.
Problemas:	F_6 *Schaffer's* com dimensão $n = 1, 2, 3, 4, 5, 6$ e $n = 60$.
Análise:	ANOVA, superfície de resposta.

Adenso-Díaz e Laguna (2006) (Tabela 5.5) apresentam um método para encontrar os melhores valores para no *máximo cinco* parâmetros de uma metaheurística. Por usar um planejamento fatorial fracionário de *Taguchi* acoplado com um procedimento de busca local, não garante que os melhores valores encontrados são os valores ótimos. Esse procedimento foi implementado em um software chamado CALIBRA. O método proposto utiliza o planejamento fatorial fracionário de *Taguchi*, um vetor ortogonal

de $L_9(3^3)$ que pode conter até quatro parâmetros com três valores críticos executando somente nove tratamentos. A não garantia de encontrar valores ótimos é minimizado pelo fato de CALIBRA usar essa análise como uma diretriz na busca através do espaço de parâmetros. Além da limitação de ajustar no máximo cinco parâmetros, o método CALIBRA não examina as interações. Nos testes iniciais com o AGP, foram encontradas várias interações significativas entre os parâmetros, como é mostrado no Capítulo 7. Outra limitação é o fato de não serem considerados os parâmetros cujos valores são qualitativos.

Tabela 5.5: *Resumo da análise experimental apresentada por Adenso-Díaz e Laguna (2006).*

Resumo	
Objetivo:	Método automático de ajuste de parâmetros — CALIBRA.
Fatores:	máximo cinco fatores quantitativos.
Medida de desempenho:	configurável: qualidade da solução ou tempo de execução.
Blocagem:	—
Planejamento:	Planejamento Fatorial Fracionário de Taguchi.
Réplicas:	configurável.
Critério de parada:	esgotar recurso predefinido.
Algoritmo:	Heurísticas diversas: busca tabu, *simulated annealing* e outras.
Parâmetros fixos:	—
Problemas:	Problema de Steiner, escalonamento de máquinas, e outros.
Análise:	Superfície de resposta.
Observações:	Não analisa interações entre parâmetros; limitado a cinco fatores quantitativos.

Em (Adenso-Díaz e Laguna, 2006) a abordagem é baseada na metodologia da superfície de resposta (MSR) (Montgomery, 2009). Esse método, que é parecido com um método do gradiente descendente ou uma busca local, consiste na busca interativa no espaço dos parâmetros que pode ser descrito da seguinte maneira: uma métrica é definida sobre o espaço de parâmetros o qual atribui um *distância* entre cada par de configurações de parâmetros da metaheurística a ser ajustada. A busca inicia-se em algum ponto do espaço de parâmetros, isto é, para um dado valor dos parâmetros, e move-se iterativamente no espaço de parâmetros considerando a sequência de pontos. A cada iteração do processo de busca, a metaheurística é testada para o valor dos parâmetros correspondentes ao ponto atual e para os dos pontos vizinhos correspondentes. Se os resultados observados no ponto atual são melhores que os dos vizinhos, a busca termina e os valores dos parâmetros correspondentes ao ponto atual são retornados. Caso contrário, o ponto atual é movido para o melhor ponto da vizinhança avaliada e o processo é iterado. O esquema de busca descrito não garante que o valor ótimo global

dos parâmetros será encontrado, mas ele encontra boas configurações, em geral. A maior desvantagem dessa abordagem está no fato de que é preciso definir uma métrica no espaço dos parâmetros. Não é problema quando os parâmetros são quantitativos, mas quando têm-se parâmetros qualitativos, seus valores não possuem uma ordem significativa. Infelizmente, o ajuste de metaheurísticas, com frequência, tem que lidar com parâmetros desse tipo, por exemplo, os tipos de seleção de indivíduos em um AG – torneio, roleta, aleatória – não apresentam nenhuma *distância* significativa entre eles.

Shahsavar et al. (2011) (Tabela 5.6) apresentam uma metodologia que é dividida em etapas, onde numa primeira etapa os parâmetros do AG assim como suas interações são analisadas estatisticamente. Depois os valores ótimos dos parâmetros são encontrados. A metodologia proposta para o projeto de um AG robusto apresenta três fases:

- **Fase 1**: Escolhas do projeto do AG

  O projeto do AG envolve a definição da codificação da solução do problema, o método de seleção, os operadores de cruzamento e mutação e o critério de parada. Os autores utilizam o termo *padrão* para denotar essas características do projeto do AG e mostram que os parâmetros do AG estão associados aos padrões escolhidos, ou seja, a escolha desses padrões influencia o desempenho do AG.

- **Fase 2**: Busca das combinações significativas dos padrões escolhidos

  O desempenho do AG é afetado por alguns fatores controláveis como as escolhas dos padrões de codificação, seleção, cruzamento, mutação, preposição e critério de parada, assim como por fatores não-controlados. As diferentes combinações desses fatores resultam em desempenhos diferentes, e um planejamento de experimentos pode ser implementado para estudar os efeitos dos fatores. Além disso, quando o estudo envolve a combinação de dois ou mais fatores, o planejamento fatorial é, em geral, o método mais indicado (Montgomery, 2009).

  Os efeitos dos diferentes padrões e parâmetros com suas interações podem ser analisados através de um planejamento fatorial como o fatorial completo 2^k ou fracionário 2^{k-p}, em que k é o número de fatores, p representa a fração do planejamento e são considerados dois níveis para cada fator. Se os resultados da análise de variância indicam significância estatística, então uma análise *post hoc* pode ser realizada para estimar a importância dos fatores. O método de Fisher da mínima diferença significativa (em inglês, LSD, *least significance difference*)

(Montgomery, 2009) e o teste de Duncan estão entre os testes que comparam todas as médias dos fatores pareados e identificam os fatores importantes .

- **Fase 3**: Ajuste dos parâmetros das combinações de padrões significativas.
  O método da superfície de resposta (MSR) – em inglês, RSM – é uma coleção de técnicas estatísticas e matemáticas úteis na modelagem e análise de problemas em que respostas são afetadas por várias variáveis, e o objetivo é otimizar essas respostas. Na maioria dos problemas, a relação entre a resposta e os fatores (variáveis independentes) não é definitiva, o que requer a estimativa dessa relação, em primeiro lugar. Para isso, se não existe curvatura na superfície de resposta, pode-se aplicar o modelo de primeira ordem; caso contrário, um modelo de mais alta ordem deve ser usado, como, por exemplo, um modelo de segunda ordem. Então, o nível de combinação entre os fatores deve ser encontrado de modo que o valor ótimo da superfície de resposta é encontrado. Para tal, o método de descida/subida mais íngreme é empregado para se movimentar no caminho de maior inclinação que faz com que a função objetivo aumente ou diminua.

  Para ajustar os parâmetros do AG pode se usar a MSR. Se a relação entre a eficiência do algoritmo e os algoritmos importantes é linear, então, dependendo das restrições (tempo e custo) sobre o tamanho do experimento pode ser usado um planejamento fatorial 2^k ou 2^{k-p}. No caso da relação não-linear, o planejamento composto central (PCC) ou outro planejamento pode ser aplicado para estimar a função de resposta. A superfície de resposta para o projeto de um AG robusto pode ser definido como: (a) minimização da porcentagem de desvio relativo – em inglês RDP – obtido pelo emprego do algoritmo para o resultado ótimo, e (b) minimização dos tempos de processamento ou otimização de outro objetivo definido pelo projeto do algoritmo. No caso de múltiplos objetivos, métodos de otimização multiobjetivo podem ser aplicados, como o trabalho apresentado em (Carrano et al., 2011).

Pontos a ressaltar sobre o trabalho de Shahsavar et al. (2011): (1) ignora que o ajuste de parâmetros depende do problema que o AG resolve; (2) não trata nem isola o ruído causado pela semente do gerador pseudo-aleatório; (3) não explica como escolheu os níveis na Fase 1 do estudo de caso; e (4) não explica as escolhas de parâmetros da Fase 2 para serem ajustados.

Tabela 5.6: *Resumo da análise experimental apresentada por Shahsavar et al. (2011).*

Resumo	
Objetivo:	Avaliar a influência dos parâmetro de um AG e encontrar a configuração ótima.
Fatores:	Oito fatores: seleção, cruzamento, mutação, reposição, critério de parada, taxa de mutação, taxa de cruzamento e taxa de melhoria local.
Medida de desempenho:	Porcentagem de desvio relativo (RDP) para solução ótima.
Blocagem:	nenhuma.
Planejamento:	2^{8-4}_{IV} fatorial fracionário.
Réplicas:	30 réplicas e teorema do limite central.
Critério de parada:	é um dos fatores analisados na Fase 1.
Algoritmo:	AG.
Problemas:	30 instâncias do problema de escalonamento.
Análise:	ANOVA, teste de Duncan, 2^4 CCF com 8 corridas axiais, superfície de resposta, multiobjetivo (precisão e eficiência de solução).
Observações:	Ignora que o ajuste depende do problema; não explica a escolha dos fatores; não trata o ruído do gerador pseudo aleatório.

Os trabalhos de Ridge e Kudenko (2007a) e Ridge e Kudenko (2007b), mostrados na Tabela 5.7, apresentam a aplicação do planejamento e análise de experimentos no estudo da metaheurística de Colônia de Formigas aplicado ao problema de roteamento.

Os autores Pinho et al. (2007) apresentam trabalho, resumido na Tabela 5.8, que tem como objetivo do projeto de experimentos determinar a escolha dos parâmetros dos AG que dará melhor precisão à resposta. O planejamento é utilizado apenas na fase de varredura dos parâmetros. É encontrada uma interação significativa com três fatores (número de gerações, taxa de cruzamento, taxa de mutação), porém não apresenta uma interpretação do que representa esse efeito.

Em Hutter et al. (2009), os autores usam a metaheurística de busca local iterativa no espaço de parâmetros. O método para configuração automática de parâmetros é denominado *ParamILS*. O método executa a busca em várias instâncias de um problema, variando um conjunto de parâmetros qualitativos e quantitativos. Estatísticas em torno da qualidade das soluções encontradas definem a função objetivo, e alguns limitantes de tempo são adicionados para manter o tempo computacional do algoritmo tratável.

O trabalho de Petrovski et al. (2005) apresenta o planejamento fatorial fracionário para a configuração de um AG na resolução do problema de escala de quimioterapia (ver Tabela 5.9). Apenas os efeitos principais são considerados, apesar de relatar o efeito

Tabela 5.7: *Resumo da análise experimental apresentada por Ridge e Kudenko (2007a) e Ridge e Kudenko (2007b).*

Resumo		
	(Ridge e Kudenko, 2007a)	(Ridge e Kudenko, 2007b)
Objetivo:	Explorar fatores que afetam a heurística colonia de formigas.	Explorar fatores que afetam a heurística colonia de formigas tipo Min-Max.
Fatores:	12 fatores: 10 do algoritmo e dois do problema, três qualitativos.	12 fatores: 10 do algoritmo e dois do problema, um qualitativo
Medida de desempenho:	Tempo de execução, qualidade da solução e erro relativo.	Tempo de execução e erro relativo
Blocagem:	Nenhuma.	Nenhuma
Planejamento:	2_{IV}^{12-5} fatorial fracionário.	Fatorial fracionário resolução V.
Réplicas:	Oito réplicas .	
Critério de parada:	Estagnação em 250 iterações.	Estagnação em 250 iterações
Algoritmo:	ACO - colônica de formigas.	ACO MMS
Problemas:	30 instâncias do problema TSP.	Instâncias produzidas por gerador automático.
Análise:	ANOVA, PCC com 24 pontos centrais.	ANOVA, PCC com 6 pontos centrais.

Tabela 5.8: *Resumo da análise experimental apresentada por Pinho et al. (2007).*

Resumo	
Objetivo:	Escolha de parâmetros dos algoritmos genéticos produzem resposta com melhor precisão.
Fatores:	Tamanho da População, Número de Gerações, Taxa de cruzamento e taxa de mutação.
Medida de desempenho:	Precisão da solução obtida.
Blocagem:	—
Planejamento:	Planejamento Fatorial Completo 2^4.
Réplicas:	Três réplicas.
Critério de parada:	—
Algoritmo:	AG representação binária.
Parâmetros fixos:	—
Problemas:	Função de teste com vários máximos locais.
Análise:	Cálculo dos efeitos.
Observações:	Realiza apenas a fase de varredura; encontra interação com três fatores mas não apresenta a interpretação desse efeito.

de interações de segunda ordem significativas em figura apresentada no seu texto.

Tabela 5.9: *Resumo da análise experimental apresentada por Petrovski et al. (2005).*

Resumo	
Objetivo:	Escolha de parâmetros dos algoritmos genéticos para melhor eficiência da solução do problema de escala de quimioterapia.
Fatores:	Oito fatores: qualitativos e quantitativos.
Medida de desempenho:	Precisão da solução obtida.
Blocagem:	semente gerador pseudo-aleatório
Planejamento:	Planejamento Fatorial Fracionário 2^{8-2}.
Réplicas:	30 réplicas.
Critério de parada:	máximo número de gerações.
Algoritmo:	AG representação binária e inteira.
Problemas:	Escala de quimioterapia.
Análise:	Efeitos, ANOVA, superfície de resposta, PCC
Observações:	Não mostra níveis dos fatores; desconsidera as interações.

5.6 Considerações do Capítulo

Neste capítulo foram apresentadas diferentes abordagens para o problema de configuração de metaheurísticas. Trabalhos relacionados com os objetivos deste trabalho, isto é, o estudo de parâmetros utilizando o planejamento e análise de experimentos foram descritos. Não foram encontrados trabalhos que aplicam o planejamento de experimentos na avaliação de Algoritmos Genéticos Paralelos em plataformas *multicore*.

Além dos métodos descritos, existem métodos automáticos para o ajuste de parâmetros de código livre, como: Mobat (`http://sourceforge.net/projects/mobat`), Bonesa (`http://sourceforge.net/projects/tuning`), e SPOT (`http://cran.r-project.org/web/packages/SPOT/index.html`).

Capítulo 6

Metodologia de Experimentação com Algoritmos

Este Capítulo apresenta uma metodologia de experimentação com algoritmos baseada nos trabalhos relevantes que tratam da análise experimental de metaheurísticas e dos métodos utilizados na execução e relato de experimentos computacionais. A partir da revisão desses trabalhos, é proposto um método para avaliação da influência de fatores no desempenho dos algoritmos evolutivos paralelos, através da aplicação do planejamento estatístico fatorial 2^k.

6.1 Introdução

De acordo com Hooker (1994), para alguns pesquisadores que trabalham com o estudo de algoritmos, apenas a análise teórica pode ser considerada como ciência, e o trabalho empírico seria algo *inculto* e *não sofisticado*. Entretanto, nem sempre os resultados obtidos pela análise teórica são suficientes para explicar o funcionamento de algoritmos na resolução de problemas reais. A análise experimental pode auxiliar nessa tarefa.

O estudo teórico de algoritmos utiliza métodos dedutivos matemáticos que caracterizam o desempenho do algoritmo em função do tamanho do problema. Os resultados da análise da complexidade do algoritmo são relatados em termos do melhor, pior e caso médio, sem entrar em detalhes do sistema físico no qual o algoritmo é executado. Alguns pontos a respeito da análise teórica são relevantes para esclarecer o papel da análise experimental no contexto do estudo de metaheurísticas:

- Estudar algoritmos de modo teórico implica em usar alguma forma de reducionismo. Reducionismo funciona em alguns casos e falha com frequência no estudo de algoritmos (Hooker, 1994).

- Algumas metaheurísticas são algoritmos estocásticos e nem sempre chegam a uma solução ótima, mesmo depois de longo tempo de execução. Essas características tornam a análise teórica difícil, e a maioria dos estudos com metaheurísticas são realizados de forma experimental (Chiarandini et al., 2007).

- Atualmente existe enorme diversidade de plataformas computacionais que não cabem em um único modelo de sistema físico. A experimentação de algoritmos em computadores atuais é relevante e necessária para obter prognósticos mais precisos sobre o seu desempenho e robustez (Bartz-Beielstein e Preuss, 2010).

A experimentação é utilizada em diversas áreas do conhecimento. Técnicas de planejamento experimental foram criadas por R. A. Fisher em 1920 para experimentação na agricultura, e tem sido aplicadas extensivamente em pesquisas das ciências naturais e em processos industriais (Spall, 2010). A experimentação com algoritmos precisa construir essa *cultura da experimentação*, com técnicas de laboratório para guiar o desenvolvimento de experimentos com rigor científico (McGeoch, 1996). Segundo Johnson (2002), a falta de rigor científico nos primeiros trabalhos experimentais direcionou Knuth e outros pesquisadores a se voltar para a análise teórica na década de 1960.

As abordagens teóricas e experimentais do estudo de algoritmos são complementares. O distanciamento entre a análise teórica e a experimental que ocorre na área da computação não ocorre em outras áreas das ciências naturais – física, biologia, química – onde os pesquisadores teóricos colaboram com os da área experimental ao submeter suas teses para serem testadas, e os experimentalistas inspiram novas teorias (Bartz-Beielstein e Preuss, 2010).

No campo das metaheurísticas e dos algoritmos evolutivos (AE), a ligação com a teoria é ainda mais fraca. Há um sentimento de que não existe uma teoria para ser testada, mas apenas ideias intuitivas sobre o que tornaria mais fácil resolver um dado problema. Grande número de pesquisadores dessa área transferem suas ideias para algoritmos. Esses algoritmos são testados antes de se realizar uma análise teórica. Assim, ocorre um processo de aprendizagem a partir da experimentação (Bartz-Beielstein e Preuss, 2010).

Hooker (1994, 1995) foi um dos primeiros a ressaltar a importância de tornar a experimentação de algoritmos em uma *ciência*, isto é, realizar experimentos com planejamento, com métodos rigorosos para a análise dos dados obtidos, e com resultados que possam ser reproduzidos. McGeoch (1996) apresenta um dos primeiros trabalhos a contribuir para o desenvolvimento de uma cultura de pesquisa experimental de algoritmos. A autora apresenta um conjunto básico de referências às técnicas estatísticas apropriadas para a experimentação de algoritmos, e apresenta conselhos sobre o que evitar.

Outros trabalhos relevantes nas áreas de computação, metaheurística e AE –como, (Barr et al., 1995; Czarn et al., 2004b; Rardin e Uzsoy, 2001)– apresentam métodos experimentais que envolvem o uso de estatística. Ainda assim, somente uma minoria dos trabalhos publicados nessas áreas usam ferramentas estatísticas. Esse fato pode ser explicado por dois fatores. Primeiro, existe a necessidade de conhecimentos de estatística e técnicas de planejamento experimental para a sua aplicação correta e para a compreensão dos resultados obtidos. Segundo, as suposições necessárias para a aplicação dos métodos estatísticos deixam os pesquisadores da ciência da computação céticos em relação à aplicação real desses métodos na experimentação com algoritmos (Bartz-Beielstein e Preuss, 2010).

Métodos estatísticos são usados para nos ajudar a entender a variabilidade. Variabilidade ocorre quando sucessivas observações de um sistema ou fenômeno não produzem exatamente o mesmo resultado. A estatística nos fornece uma estrutura para descrever essa variabilidade, para aprender sobre quais fontes potenciais de variabilidade são mais importantes, se há interações entre elas e quais as influências dessas interações (Montgomery e Runger, 2009).

6.2 Terminologia e Notação

De acordo com McGeoch (1996), no experimento computacional, o objeto da análise não é o algoritmo abstrato, mas sim programas de aplicação e programas de simulação para resolver instâncias de problemas executados em um determinado ambiente computacional. *Programas de simulação* diferem de *programas de aplicação* por proporcionarem ao pesquisador o controle completo do ambiente experimental. Detalhes de implementação podem ter um certo impacto no desempenho dos programas derivados dos algoritmos em estudo. Esses detalhes são omitidos em uma análise teórica.

McGeoch distingue diferentes *níveis de instanciação* para os algoritmos analisados. Metaheurísticas são exemplos de algoritmos minimamente instanciados que contém poucos detalhes de implementação (Bartz-Beielstein e Preuss, 2010). Um algoritmo com maior nível de instanciação inclui detalhes básicos da implementação, se utiliza pilha ou fila para uma dada estrutura de dados. Um algoritmo altamente instanciado incluiria detalhes específicos de uma linguagem de programação em particular ou de uma dada arquitetura de computador.

Ao longo deste texto, o termo *problema* é usado para denotar um problema genérico, por exemplo, o do caixeiro viajante ou o do escalonamento de tarefas, e uma *instância do problema* corresponde a uma representação específica das características do problema, como o tamanho do problema, número e valores de restrições. No caso do problema do caixeiro viajante uma instância é definida pelo número de pontos que devem ser visitados e suas restrições.

Para Bartz-Beielstein (2006), ao caracterizar a instância de um problema são dados valores ao conjunto de parâmetros do problema (*design do problema*). Da mesma forma, existe o conjunto de parâmetros do algoritmo (*design do algoritmo*) que pode ser um conjunto vazio ou de tamanho variável. Por exemplo, um *design do algoritmo* inclui parâmetros como o tamanho da população e um *design do problema* compreende as dimensões do espaço de busca, pontos iniciais ou função objetivo.

6.3 O Experimento

Um experimento é um conjunto de execuções de testes sob condições controladas e com um propósito definido. Segundo Neto et al. (2010) "*o que leva o pesquisador a fazer experimentos é o desejo de encontrar a solução de determinados problemas*". O experimento possibilita a comprovação de uma teoria, a revelação de conhecimentos sobre um processo e o dimensionamento do efeito de um ou mais fatores sobre algum fenômeno. Um fator é qualquer variável controlável em um experimento, que influencia o seu resultado ou sua resposta.

O experimentador tem uma gama de decisões a tomar sobre: a seleção de problemas a serem resolvidos pelo algoritmo, o nível de instanciação do algoritmo, o ambiente computacional, medidas de desempenho, parâmetros do problema, parâmetros do algoritmo e relato dos resultados. Cada escolha realizada tem um efeito substancial e significativo nos resultados e na significância do experimento.

De acordo com Neto et al. (2010), a atividade mais importante da estatística não é a análise de dados, e sim o planejamento dos experimentos em que esses dados devem ser obtidos. Quando o planejamento não é feito de forma apropriada, o resultado muitas vezes é um aglomerado de números estéreis, sem significado, do qual é pouco provável que seja possível extrair qualquer conclusão.

Para Montgomery (2009), o uso de uma abordagem estatística no planejamento e análise de um experimento somente é possível se todos envolvidos tem uma ideia clara antecipadamente do que vai ser estudado, como os dados serão coletados e um entendimento de como esses dados serão analisados. Esse processo foi adaptado por Barr et al. (1995) em etapas para a experimentação de heurísticas. Essas etapas são mostradas a seguir:

1. Definir os objetivos do experimento.
2. Escolher medidas de desempenho e fatores a explorar.
3. Planejar e executar o experimento.
4. Analisar os dados e extrair conclusões.
5. Relatar os resultados.

As próximas seções descrevem essas etapas com maiores detalhes.

6.4 Objetivos da Experimentação

O experimento deve ter um propósito claro e definido antes de iniciar os testes. A experimentação com metaheurísticas tem sempre implícita nos seus objetivos a avaliação da relação que motivou a sua origem: a troca da qualidade da solução por um tempo de execução menor.

Barr et al. (1995) argumenta que uma heurística traz alguma contribuição se ela é:

- Rápida - produz soluções de alta qualidade mais rápido do que outros algoritmos;
- Precisa - identifica soluções de melhor qualidade do que outros algoritmos;
- Robusta - menos sensível a diferenças nas características dos problemas, qualidade dos dados e aos parâmetros ajustáveis do que outros algoritmos;
- Simples - fácil de implementar;

- Alto impacto - resolve problemas novos de forma rápida e precisa;
- Geral - aplica-se a um grande conjunto de problemas;
- Inovadora - novo e criativo.

Qualquer que seja a contribuição da heurística, a experimentação será necessária para confirmar o que o autor afirma que ela faz e a comprovação deve ser feita por testes estatísticos.

6.4.1 Classificação da Experimentação

Segundo Bartz-Beielstein (2006), as razões para se fazer um experimento se resumem em: (1) *Eficácia* – comparar o desempenho de algoritmos diferentes para as mesmas instâncias de problemas; e (2) *Eficiência* – caracterizar o desempenho de um algoritmo. A *eficácia* está relacionada com o objetivo de provar que o algoritmo estudado resolve diferentes instâncias do problema, e de comparar o seu desempenho com o de outros algoritmos. Para estudar a *eficiência*, busca-se uma configuração do conjunto de parâmetros do algoritmo que melhore o seu desempenho na resolução de uma mesma instância do problema.

McGeoch (1996) classifica a experimentação segundo os objetivos almejados pelo pesquisador em: (1) estudo de dependência – *dependency study*; (2) estudo da robustez – *robustness study*; e (3) estudo de sondagem – *probing study*. O estudo de dependência tem como objetivo descobrir o desempenho médio do algoritmo e evidências de quais e como os fatores influenciam as medidas de desempenho. O estudo de robustez implica em mais execuções aleatórias do algoritmo para determinar as propriedades da distribuição dos dados coletados. O estudo de sondagem investiga nuances do funcionamento do algoritmo que podem afetar o seu desempenho.

Para Rardin e Uzsoy (2001), a investigação experimental de heurísticas é conduzida por diversas razões, e a maneira que ela deve ser conduzida depende do contexto da experimentação. Segundo os autores, existem três contextos mais importantes, que são: (1) *pesquisa vs. desenvolvimento*; (2) *projeto vs. planejamento vs. aplicações de controle*; e (3) o *ciclo de vida para o problema em estudo*.

No primeiro contexto aparecem os testes de *pesquisa* que são direcionados para a descoberta de novas heurísticas para problemas existentes ou a aplicação de heurísticas existentes de forma criativa a novos problemas. Procura-se conhecer o que funciona, o

que não funciona, e o porquê de ser assim, até o nível onde os conhecimentos adquiridos são transferíveis para além do domínio do programa específico sob estudo. Testar na fase de *desenvolvimento* da heurística envolve o procedimento de solução mais eficiente para um ambiente específico. Por exemplo, na adaptação de um algoritmo conhecido de escalonamento de máquinas para uma determinada fábrica, o foco está na implementação do algoritmo e no ajuste dos seus parâmetros para produzir os melhores resultados para a aplicação.

O segundo contexto trata das tarefas de otimização. Esse contexto difere do anterior quanto ao tempo disponível para se encontrar uma solução e quanto à importância de se obter a resposta correta. Os *problemas de projeto – design problems* – são aqueles que são resolvidos com pouca frequência e buscam por respostas que podem levar muito tempo para serem encontradas. São problemas em que o crítico é a qualidade da solução, como o dimensionamento de uma rede de telecomunicações,

No outro extremo do segundo contexto, encontram-se os *problemas de controle – control problems* – que devem ser resolvidos com frequência e envolvem decisões que cobrem um relativo curto período de tempo. Algoritmos devem ser o mais rápido possível, e a qualidade da solução não é tão importante, como na transmissão de dados multimídia através de uma rede de computadores.

Ainda no segundo contexto, estão os *problemas de planejamento – planning problems* – que ocupam uma posição intermediária em termos de frequência e qualidade da solução, como na definição de um calendário de encontros e escala de turnos.

No terceiro e último contexto – do ciclo de vida do problema – dependendo do estágio em que o algoritmo se encontra, os objetivos diferem. Um algoritmo, proposto para solucionar um problema para o qual não havia solução prática, tem o objetivo inicial de mostrar que é capaz de resolver o problema. Numa próxima etapa, quando já se sabe que o algoritmo é capaz de resolver o problema, o objetivo passa a ser de comparação com outros algoritmos, para verificar qual tem melhor desempenho e para verificar o desempenho ao resolver uma ampla variedade de instâncias de problema.

6.4.2 Instâncias de Teste

Um dos grandes desafios na condução de experimentos computacionais é reunir ou construir as instâncias de teste na quantidade suficiente, com tamanho e variedade para abranger todas as características do problema que são de interesse do experimento.

Para Rardin e Uzsoy (2001), as instâncias de problemas podem ser classificas em:

1. Dados provenientes de situações reais (instâncias reais).
   São em geral os melhores conjuntos de instâncias, mas pode ser difícil conseguilos. A coleta de dados reais pode consumir tempo e esforço, que o experimentador não pode dispor.

2. Variações de instâncias reais.
   Uma alternativa é substituir valores reais por valores aleatórios. A estrutura da instância real é preservada, e alguns valores numéricos são modificados. Isso pode tornar uma instância mais complexa.

3. Bibliotecas públicas de referência.
   São muito utilizadas e existem muitos repositórios de instâncias para muitos problemas clássicos, inclusive com instâncias reais, como por exemplo, OR–Library[1], TSPLib[2], NetLib[3].

4. Instâncias geradas aleatoriamente.
   São instâncias geradas aleatoriamente a partir de uma semente e uma lista de parâmetros, como o tamanho e detalhes da estrutura da instância a ser gerada. A principal vantagem do uso de um gerador de instâncias é que ele possibilita que um estudo sistemático seja realizado: a cada inclusão de alguma característica específica à instância, é verificado como o algoritmo se comporta.

Johnson (2002) alerta que as instâncias geradas aleatoriamente podem levar a conclusões distorcidas em relação ao mundo real. Muitas vezes elas retratam situações muito mais difíceis de resolver do que os problemas reais. Sempre que as instâncias são geradas pelo pesquisador, o processo de geração deve ser claramente descrito para ser possível que outros a reproduzam.

Segundo Eiben e Jelasity (2002), para que os resultados produzidos pela análise experimental possam ser generalizados, isto é, gerar conhecimento científico a partir de dados empíricos por indução, é necessário não apenas aumentar o número de problemas a serem testados mas também estabelecer *classes de problemas*. Problemas pertencentes

[1]http://http://people.brunel.ac.uk/ mastjjb/jeb/info.html
[2]http://http://comopt.ifi.uni–heidelberg.de/software/TSPLIB95/
[3]http://www.netlib.org

a uma dada classe são similares e diferentes dos pertencentes a outra classe. Os autores mostram que não existe uma classificação padrão das classes.

6.5 Medidas de Desempenho

Algoritmos são frequentemente comparados pelos seus tempos de execução e pela qualidade da sua solução. Enquanto o tempo de execução é uma medida de interesse para qualquer algoritmo, as comparações pela qualidade da solução são em geral apropriadas somente para algoritmos de aproximação e metaheurísticas.

Para McGeoch (1996), a escolha das medidas de desempenho depende das questões que motivam a pesquisa. Barr et al. (1995) observam que essas questões podem ser categorizadas de forma ampla em questões de qualidade, esforço computacional e robustez. A robustez pode ser definida como a capacidade da metaheurística de encontrar soluções sobre uma grande variedade de instâncias de um problema ou sobre uma grande variedade de instâncias de diferentes problemas.

Bartz-Beielstein (2006) apresenta um esquema de classificação que distingue entre efetividade e eficiência. Efetividade está relacionada com a robustez e com a capacidade do algoritmo produzir o efeito desejado. Por outro lado, as medidas de eficiência procuram responder questões do tipo: o algoritmo produz os efeitos desejados sem desperdício?

Coffin e Saltzman (2000) apresentam um levantamento com outras métricas além das que são descritas nesta seção. Bartz-Beielstein (2006) também apresenta e discute medidas de desempenho além das listadas nesta seção. Rardin e Uzsoy (2001) ressaltam que se nenhuma das medidas de desempenho existentes satisfazem as necessidades do experimento pode-se criar a sua própria medida.

A seguir são apresentadas as medidas mais utilizadas na avaliação de metaheurísticas.

6.5.1 Medidas com Relação ao Tempo de Execução

As medidas que focam na velocidade com que o algoritmo consegue solucionar o problema são listadas a seguir:

Tempo de Execução

O tempo de execução corresponde à medida do tempo de processamento da metaheurística em uma dada plataforma computacional. O uso dessa medida é dos mais controversos.

Vários autores (Coffin e Saltzman, 2000; Eiben e Jelasity, 2002; Hooker, 1995; McGeoch, 1996; Rardin e Uzsoy, 2001) não recomendam o uso do tempo de execução como medida de desempenho. O tempo de execução é difícil de reproduzir, mesmo quando se utiliza a mesma configuração da plataforma computacional. Fatores como a linguagem de programação utilizada, habilidades do programador e processos em execução em segundo plano, tornam difícil reproduzir um teste e obter o mesmo tempo de execução. Outra dificuldade é dimensioná-lo para outra plataforma computacional.

O tempo de execução é uma medida que pode ser influenciada pela ordem das observações. É preciso ter o cuidado de aleatorizar a ordem de execução dos testes para minimizar essa influência.

Alguns dos testes estatísticos que podem ser aplicados na análise dos dados coletados assumem que os dados tenham uma distribuição normal. Os tempos de execução são frequentemente não-normais (Bartz-Beielstein, 2006; Bartz-Beielstein e Preuss, 2010; Chiarandini et al., 2007; Coffin e Saltzman, 2000). De acordo com Chiarandini et al. (2007), o tempo de execução de metaheurísticas frequentemente apresenta uma distribuição exponencial. Eiben e Jelasity (2002) sugerem que para medir a velocidade do algoritmo é melhor usar a quantidade de avaliações executadas até encontrar a melhor solução.

Mesmo com todos os pontos desfavoráveis ao uso do tempo de execução como medida de desempenho, Johnson (2002) defende que ele seja coletado e relatado em todos os experimentos, pois pode trazer mais informações ao leitor. Pode dar ao leitor uma ideia se o tempo de execução é competitivo ou se um algoritmo é claramente mais rápido do outro.

A natureza estocástica dos AE introduz uma variabilidade aleatória na resposta produzida pelo algoritmo: a solução obtida pode variar entre uma execução e outra, e mesmo quando a mesma solução é obtida, o esforço computacional requerido é, em geral, diferente entre execuções do mesmo algoritmo. Nesse caso, têm-se duas possíveis medidas de desempenho: qualidade da solução e esforço computacional. Em alguns

casos, quando a convergência pode ser assegurada, seria possível considerar o esforço computacional como o único indicador de desempenho relevante para o algoritmo.

Na avaliação de desempenho de computadores tradicional, Hennessy e Patterson (2006) consideram o tempo de execução real dos programas como a única medida de desempenho consistente e confiável. Tempos de execução têm sido continuamente atrapalhados pela variabilidade do desempenho do computador, em especial no caso de programas paralelos que são afetados por concorrências no acesso aos dados, escalonamento de processos, mecanismos de sincronização, disputas por recursos compartilhados, entre outros. Mazouz et al. (2011) confirmam que os processadores *multicore* trazem ainda mais variabilidade aos tempos de execução.

Os tempos de execução podem ser definidos de maneiras diferentes, dependendo do que é contabilizado. A definição mais direta de tempo é chamada de tempo *wall clock*, que consiste no tempo decorrido para completar uma tarefa, incluindo o tempo de acesso a disco, acesso a memória, atividades de entrada e saída, dentre outros. Vários termos denotam esse tempo, tais como, tempo de resposta, tempo decorrido e tempo total de processamento.

Na computação paralela, o tempo total de processamento é utilizado em uma fórmula chamada *speedup*. O *speedup* é a medida mais utilizada na avaliação do desempenho paralelo.

Quantidade de Avaliações

A quantidade de avaliações corresponde ao número de avaliações executadas pela metaheurística. Essa medida fornece um bom indicativo do custo computacional do algoritmo, se for considerado que todas as avaliações gastam a mesma quantidade de tempo, e que as avaliações consomem a maior parte do tempo de execução do algoritmo. Entretanto, quando algumas avaliações demoram mais do que outras, as quantidade de avaliações dos dois algoritmos seriam próximos, mas um algoritmo seria mais lento do que o outro. Quando um algoritmo usa alguma técnica de reparo chamada pela rotina de avaliação, um algoritmo em que o reparo é chamado com frequência terá avaliações que demoram mais do que outro algoritmo em que pouco reparo é necessário. Outro caso em que a quantidade de avaliações pode não ser representativo do custo computacional é quando o tempo gasto com a avaliação é muito pequeno, e outros componentes do algoritmo possuem um grande impacto no tempo de execução (Eiben

e Jelasity, 2002).

Quantidade de Gerações

É uma métrica muito utilizada em AE e é simples de interpretar: quanto menor a quantidade de gerações melhor o seu desempenho computacional. Porém, na comparação entre experimentos publicados pode não existir uma relação direta com o número de avaliações por geração, uma vez que o tamanho da população pode ser diferente. Além disso, o tamanho da população pode variar ao longo das gerações para um mesmo algoritmo. A quantidade de gerações é quase sempre inferior à quantidade de avaliações, no melhor caso são equivalentes.

Essa métrica pode não ser válida para comparação de algoritmos distintos, mas poderia ser usada na comparação de duas configurações para uma mesmo algoritmo, como no ajuste de parâmetros do algoritmo.

6.5.2 Medidas com Relação à Qualidade da Solução

Para Rardin e Uzsoy (2001), o maior desafio da análise experimental é avaliar a qualidade das soluções encontradas por metaheurísticas. As metaheurísticas buscam solução para problemas difíceis (*NP-difícil*) e, muitas vezes, os métodos exatos não produzem soluções de qualidade em tempo viável. Algumas formas de contornar esse problema e conseguir avaliar a qualidade da solução podem ser: (1) cálculo da solução exata para instâncias de tamanho pequeno; (2) uso de limites inferiores e superiores; (3) construção de instâncias a partir de valores ótimos conhecidos; (4) estimativa estatística de valores ótimos conhecidos; e (5) comparação com os melhores valores conhecidos.

As principais medidas que buscam dimensionar a qualidade da solução encontrada são listadas a seguir.

Melhor Solução Encontrada

É uma medida que aparece com frequência nas publicações de experimentos com metaheurísticas – em inglês, *best–of–n*. Pode ser útil quando se trata de um problema de projeto (seção 6.4.1), uma vez que somente a solução final será implementada (Eiben e Jelasity, 2002).

Entretanto, Birattari e Dorigo (2006) criticam o uso dessa métrica nos experimentos com algoritmos estocásticos, com o argumento de que não é de qualquer interesse real, pois trata-se de uma medida superestimada do desempenho. Ela indica ser prove-

niente de um experimento de avaliação do desempenho de uma versão aleatoriamente reiniciada n vezes. Essa reinicialização aleatória que consiste em repetir a execução do mesmo algoritmo, sem nenhuma melhoria ou entrada da execução anterior, é tão trivial que não seria uma estratégia de reinicialização sensata de se usar. Johnson (2002) também critica essa medida porque a melhor solução é uma amostra da cauda de uma distribuição e é, necessariamente, menos reproduzível do que quando fornecido o valor médio.

Valor Médio das Melhores Soluções Encontradas

O valor médio das melhores soluções encontradas – em inglês, *mean best fitness (MBF)* – é uma medida fácil de se obter, porém apresenta os seguintes pontos desfavoráveis: (1) a média não traz informações sobre a distribuição dos resultados, ou seja, um algoritmo estável com desempenho medíocre pode ser equiparado pela média a um algoritmo não–estável que encontra soluções boas, mas falha muitas vezes; e (2) quando a solução ótima global não é conhecida, fica difícil interpretar as diferenças obtidas: "Um aumento de 1% significa muito ou não?"(Bartz-Beielstein e Preuss, 2010)

Taxa de sucesso (TS)

A taxa de sucesso (TS) informa quantas vezes a solução foi encontrada em várias repetições da execução do algoritmo. A execução do algoritmo ocorre até um limite pré-definido de esforço computacional, como por exemplo, um número máximo de avaliações.

Essa medida possibilita medir a robustez da metaheurística em termos de porcentagem de sucesso. Entretanto, está limitada a instâncias de problema para as quais a solução ótima é conhecida. Outra desvantagem é a necessidade de se determinar qual seria o limite do custo computacional adequado. Suponha que um algoritmo A_1 necessite de pelo menos 10^6 avaliações para chegar à solução e que o algoritmo A_2 precise de apenas 10^5 avaliações. Se for considerado o valor do máximo de avaliações como sendo 10^7, os dois algoritmos teriam 100% de acerto e seriam considerados iguais.

6.5.3 Medidas para Algoritmos Paralelos

Barr e Hickman (1993) apresentam uma discussão sobre medidas de desempenho para algoritmos paralelos. Várias métricas da eficiência de implementações paralelas resultam em dificuldades de medição e comparação, decorrentes das limitações impos-

tas pelas diferentes arquiteturas das máquinas, os diferentes projetos de máquinas, a natureza estocástica de alguns algoritmos paralelos e as oportunidades inerentes para a introdução de vieses. O *speedup* é uma medida amplamente utilizada para descrever a eficiência conseguida sobre o processamento serial pelo uso de vários processadores.

Alba e Luque (2006) argumentam que conclusões diferentes podem ser inferidas dos mesmos resultados, dependendo da medida usada e de como essa medida é aplicada. A seguir são descritas algumas medidas de desempenho para metaheurísticas paralelas.

Speedup

O *speedup* apresenta o quanto um algoritmo paralelo é mais rápido que o melhor algoritmo sequencial correspondente. Para algoritmos paralelos determinísticos, o *speedup* é dado pela razão T_1/T_m, onde m é o número de processos, T_1 e T_m são os tempos de execução do algoritmo sequencial e do algoritmo paralelo com m processos, respectivamente.

Para algoritmos estocásticos como os AE, essa definição de *speedup* não pode ser aplicada diretamente. Como os tempos de execução dos AE podem variar, é necessário repetir as execuções, e estimar o tempo a ser utilizado no cálculo do *speedup*.

O *speedup* s_m para um AEP é a razão da média dos tempos de execução com um processo $\overline{T}_1$ e da média dos tempos de execução com m processos $\overline{T}_m$, como é mostrado a seguir.

$$s_m = \frac{\overline{T}_1}{\overline{T}_m} = \frac{(\sum_{i=1}^{k} T_{1i})/k}{(\sum_{j=1}^{p} T_{mj})/p} \tag{6.1}$$

onde T_{1i} e T_{mj} correspondem aos tempos de processamento total (*wall clock time*) para cada uma das k execuções com um processo (corresponde à execução sequencial) e p execuções paralelas com m processos, respectivamente.

Os valores de s_m podem ser interpretados da seguinte maneira: (1) $s_m = 1$ significa que não houve ganho de desempenho na versão paralela; (2) $s_m = m$ significa aceleração linear; (3) $s_m > m$ aceleração superlinear; e (4) $s_m < 1$ aceleração sublinear – houve perdas com a paralelização.

Alba (2002) desenvolve definições de *speedup* baseado em como os valores de T_1 e T_m são entendidos, como mostra a Tabela 6.1.

O tipo I – *Speedup* Forte – compara o tempo de processamento paralelo contra o algoritmo sequencial mais eficiente conhecido. Essa é a definição mais exata de *speedup*,

I. *Speedup* forte
II. *Speedup* fraco
A. *Speedup* com melhor solução
1. *Versus panmixia*
2. Ortodoxo
B. *Speedup* com esforço predefinido

Tabela 6.1: *Taxonomia das medidas de speedup propostas por Alba (Alba, 2002).*

porém esse tipo quase não é usado, dada a dificuldade em se encontrar o atual algoritmo mais eficiente.

O tipo II – *Speedup* Fraco – compara o tempo do algoritmo paralelo desenvolvido por um pesquisador contra seu próprio algoritmo sequencial. Nesse caso, dois critérios de parada possíveis para o algoritmo determinam os subtipos A e B: parada ao encontrar a solução ótima ou próxima da ótima e parada por atingir um esforço predefinido.

Para o tipo II.A – *Speedup* com melhor solução – existem duas variantes: *Versus Panmixia* e Ortodoxo. O *Versus Panmixia* é quando se compara o tempo de um algoritmo paralelo contra o algoritmo sequencial canônico, ou seja, compara dois algoritmos diferentes. O Ortodoxo é quando se compara o tempo de um algoritmo paralelo rodando em um processador, com o mesmo algoritmo rodando em m processadores, isto é, o mesmo algoritmo é comparado. Neste trabalho, o *speedup* utilizado é do tipo Ortodoxo.

Alba (2002) descarta o tipo II.B – *Speedup* com esforço predefinido –, porque é contra o intuito do *speedup* comparar algoritmos que não produzam resultados de mesma acurácia.

Barr e Hickman (1993) apresentam uma taxonomia diferente: (1) *Speedup* – mede a razão entre o tempo do código serial mais rápido em uma máquina paralela e o tempo do código paralelo usando m processadores na mesma máquina paralela; (2) *Speedup* Relativa – é a razão entre o tempo de execução serial do código paralelo em um processador e o tempo de execução do código paralelo em m processadores – similar ao tipo II.A.2 – ortodoxo – da Tabela 6.1; e (3) *Speedup* Absoluto – compara o tempo serial mais rápido em qualquer máquina e o tempo paralelo em m processadores – similar ao tipo I da Tabela 6.1.

Alba et al. (2013) recomendam que o AEP deveria computar soluções com mesma precisão que o algoritmo sequencial. Essa precisão pode ser a da solução ótima, se co-

nhecida, ou uma solução aproximada, desde que ambos algoritmos produzam a solução de mesma precisão. O critério de parada dos algoritmos sequencial e paralelo deve ser encontrar a mesma solução. Os autores também aconselham a execução do mesmo algoritmo paralelo para obter os tempos sequenciais e paralelos, onde os tempos sequenciais são obtidos com a execução do AEP com apenas um processo. Desta forma, tem-se um *speedup* válido, prático (não é necessário o melhor algoritmo sequencial conhecido) e ortodoxo (mesmo código, mesma precisão).

Speedup com Medianas

Touati et al. (2013) apresentam um metodologia de avaliação de desempenho baseada no cálculo do *speedup* com as medianas dos tempos de execução observados, ao invés de usar a média. Os autores argumentam que é preciso controlar a variabilidade dos tempos de execução e a mediana é robusta a pontos discrepantes. Cabe ressaltar que o teorema do limite central não se aplica à mediana, mas se aplica à média (ver Seção 6.6.1).

Eficiência

A eficiência e_m mede a fração de tempo que um processador é de fato utilizado, e é dada pela normalização da *speedup* s_m, como mostra a Equação 6.2.

$$e_m = \frac{s_m}{m} \tag{6.2}$$

Idealmente o valor de e_m deveria ser 1, mas existe o tempo gasto para comunicação, alocação de memória, por exemplo, e o seu valor nunca chega a 1, a não ser quando ocorre *speedup* superlinear, no qual e_m passa a ser maior que 1.

Fração Serial

A fração serial mede a proporção do algoritmo que é inerentemente sequencial, como mostra a Equação 6.3:

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m} \tag{6.3}$$

Numa situação ideal, o valor da fração serial deveria se manter constante para diferentes valores de m. Se o valor de *speedup* é pequeno, dado que a perda de eficiência

é causada pelo paralelismo limitado do algoritmo, ainda assim pode se considerar o resultado como bom se f_m for constante para diferentes valores de m. Por outro lado, um aumento suave de f_m é um aviso de que a granularidade das tarefas paralelas é muito fina. E se f_m diminuir à medida que m aumentar, significa que o *speedup* está se tornando superlinear. Se ocorre uma aceleração superlinear então o valor de f_m ficará negativo.

Eficiência Incremental

A eficiência incremental (Equação 6.4) mostra a fração de melhoria do tempo pela adição de outro processador. É usada quando o tempo de um único processador é desconhecido. Essa medida foi generalizada (Equação 6.5) para medir a melhoria alcançada pelo aumento do número de processadores de n para m.

$$ie_m = \frac{(m-1) \cdot E[T_{m-1}]}{m \cdot E[T_m]} \tag{6.4}$$

$$gie_{m,n} = \frac{n \cdot E[T_n]}{m \cdot E[T_m]} \tag{6.5}$$

Speedup Escalável

As métricas anteriores mostram melhorias no desempenho proveniente da adição de elementos de processamento, porém não medem a utilização da memória disponível. A *speedup* escalável (Equação 6.6) dimensiona a utilização completa dos recursos da máquina.

$$ss_m = \frac{E[T_{1,nm}]}{E[T_{m,nm}]}, \tag{6.6}$$

onde n é o tamanho do maior problema que pode ser armazenado na memória associada a um processador, $E[T_{1,nm}]$ é o tempo de execução médio necessário para 1 processador resolver problema de tamanho nm, e $E[T_{m,nm}]$ é o tempo de execução médio necessário para m processadores resolverem problema de tamanho nm. A sua maior desvantagem é que estimar com precisão o tempo serial é difícil e é impraticável para muitos problemas (Alba, 2005).

Scaleup

A medida *scaleup* (Equação 6.7) está relacionada com a medida *speedup* escalável, porém não se baseia em uma estimativa do tempo para um processador.

$$su_{m,n} = \frac{E[T_{m,k}]}{E[T_{nm,nk}]}, \tag{6.7}$$

onde $E[T_{m,k}]$ é o tempo de execução médio necessário para m processadores resolverem problema de tamanho k, e $E[T_{nm,nk}]$ é o tempo de execução médio para nm processadores resolverem problema de tamanho nk.

6.6 Inferência Estatística

Nesta seção são apresentados conceitos básicos do campo da inferência estatística que são importantes para a escolha da medida de desempenho e para a análise correta dessas medidas.

Para tirar conclusões acerca de uma população, os métodos da estatística inferencial utilizam a informação contida em uma amostra aleatória proveniente da população. Cada valor numérico contido nos dados é o valor observado de uma variável aleatória. As variáveis aleatórias são geralmente consideradas independentes e distribuídas identicamente. Essas variáveis aleatórias são conhecidas como uma amostra aleatória. A principal finalidade em se obter uma amostra aleatória é adquirir informações sobre os parâmetros desconhecidos da população.

Uma estatística é qualquer função das observações em uma amostra aleatória. Exemplos de estatísticas são a média amostral $\overline{X}$, a variância da amostra S^2 e o desvio padrão S. Desde que uma estatística seja uma variável aleatória, ela tem uma distribuição de probabilidade chamada de distribuição amostral.

A mais importante distribuição amostral é a distribuição de probabilidades de $\overline{X}$, isto é, a distribuição amostral da média. Mesmo quando a distribuição da população, com média μ e variância σ^2, da qual estamos amostrando seja desconhecida, a distribuição amostral da média da amostra será aproximadamente normal, com média μ e variância σ^2/n, se o tamanho n da amostra for grande e as observações forem independentes. Esse é um dos mais úteis teoremas em estatística, o teorema do limite central, descrito na seção seguinte.

6.6.1 Teorema do Limite Central

Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatórias independentes com média finita μ e variância σ^2. O teorema do limite central (TLC) estabelece que a variável aleatória Sum_n definida por

$$Sum_n = \sum_{i=1}^{n} X_i \tag{6.8}$$

converge para um distribuição de dados normal $N(n\mu, n\sigma^2)$, à medida que n aproxima-se de ∞.

Em outras palavras, segundo o TLC, mesmo que a distribuição dos tempos de execução não seja normal, a distribuição da média de uma amostra de tempos de execução se aproxima de uma distribuição normal à medida que o tamanho da amostra cresce. De maneira geral, é aceito para tamanhos de amostra maiores que 30 ($n \geq 30$), a distribuição da média das amostras se aproximará de uma distribuição normal (Montgomery e Runger, 2003).

Neste trabalho, uma importante aplicação do TLC surge na estimação do *speedup* como a razão entre duas médias, $\overline{T}_1$ e $\overline{T}_m$. Se o número de execuções do algoritmo é grande o suficiente, a distribuição de $\overline{T}_m$ e $\overline{T}_1$ será próxima de uma distribuição normal, ou seja, o *speedup* é a razão entre duas variáveis aleatórias com distribuição de dados normal, e isso tem implicações importantes, como é descrito na seção seguinte.

6.6.2 Razão Entre Normais

A distribuição F_z da razão $Z = X/Y$ das variáveis aleatórias X e Y não é necessariamente normal. Se na avaliação dos AEP for adotado o *speedup* como medida de desempenho, é necessário assegurar que a função densidade de probabilidade do *speedup* se aproxime da função densidade de probabilidade normal.

Nessa seção são apresentados os resultados de Qiao et al. (2006) e Díaz-Francés e Rubio (2013) que abordam a estimação da média da razão entre duas variáveis aleatórias normalmente distribuídas.

Considere uma amostra de n observações (X, Y) de uma população bivariada normal $N(\mu_X, \mu_Y, \sigma_X, \sigma_Y, \rho)$, $\mu_x, \mu_y \neq 0$, X e Y não correlacionadas. Qiao et al. (2006) define a razão aritmética $\overline{R}_A$:

$$\overline{R}_A = \frac{\sum X_i/Y_i}{n}, \tag{6.9}$$

e a razão ponderada $\overline{R}_W$:

$$\overline{R}_W = \frac{\overline{X}}{\overline{Y}} = \frac{\sum X_i/n}{\sum Y_i/n} = \frac{\sum X_i}{\sum Y_i}. \tag{6.10}$$

Dado que $X \sim N(\mu_X, \sigma_X^2)$ e $Y \sim N(\mu_Y, \sigma_Y^2)$, segue que $\overline{X} \sim N(\mu_X, \sigma_X^2/n)$, $\overline{Y} \sim N(\mu_Y, \sigma_Y^2/n)$. O coeficiente de variação de Y é $\delta_Y = \sigma_Y/\mu_Y$ e o coeficiente de variação de $\overline{Y}$ é $\delta_{\overline{Y}} = \sigma_Y/\mu_Y\sqrt{n}$.

As simulações de Qiao et al. (2006) demonstram que, contanto que $\delta_Y < 0.2$, ambos $\overline{R}_W$ e $\overline{R}_A$ são estimadores válidos de μ_X/μ_Y. Porém, se $\delta_{\overline{Y}} < 0.2$, então $\overline{R}_W$ é um estimador aceitável de μ_X/μ_Y.

Em situações práticas, quando a média da população μ e o desvio padrão σ não são conhecidos, eles podem ser estimados pela média $\overline{Y}$ e o desvio padrão S da amostra. De acordo com Qiao et al. (2006), um estimador de tamanho de amostra n_s é dado por

$$n_s > 25(S_Y^2/\overline{Y}^2), \tag{6.11}$$

onde S_Y^2 é a variância e $\overline{Y}$ é a média da amostra.

Outra abordagem é apresentada por Díaz-Francés e Rubio (2013). Os autores demonstram a existência de uma aproximação normal à distribuição de $Z = X/Y$, em um intervalo I centralizado em $\beta = E(X)/E(Y)$, que é dado para o caso onde ambos X e Y são independentes, tem média positiva e seus coeficientes de variação cumprem as condições estabelecidas pelo Teorema 6.6.1.

Theorem 6.6.1 (Díaz-Francés e Rubio (2013)) *Seja X variável aleatória com distribuição normal com média positiva μ_X, variância σ_X^2 e coeficiente de variação $\delta_X = \sigma_X/\mu_X$ tal que $0 < \delta_X < \lambda \leq 1$, onde λ é uma constante conhecida. Para cada $\varepsilon > 0$, existe $\gamma(\varepsilon) \in \left(0, \sqrt{\lambda^2 - \delta_X^2}\right)$ e também um variável aleatória Y normal, independente de X, com média positiva μ_Y, variância σ_Y^2 e coeficiente de variação $\delta_Y = \sigma_Y/\mu_Y$ que satisfaz as condições:*

$$0 < \delta_Y \leq \gamma(\varepsilon) \leq \sqrt{\lambda^2 - \delta_X^2} < \lambda \leq 1, \tag{6.12}$$

para as quais o seguinte resultado é verdadeiro.

Qualquer z que pertence ao intervalo:

$$I = \left[\beta - \frac{\sigma_Z}{\lambda}, \beta + \frac{\sigma_Z}{\lambda}\right], \tag{6.13}$$

onde $\beta = \mu_X/\mu_Y$, $\sigma_Z = \beta\sqrt{\delta_X^2 + \delta_Y^2}$, *satisfaz*

$$|G(z) - F_Z(z))| < \varepsilon, \tag{6.14}$$

onde $G(z)$ *é a função de distribuição de uma variável aleatória normal com média* β, *variância* σ_Z^2, *e* F_Z *é a função de distribuição de* $Z = X/Y$.

O Teorema 6.6.1 estabelece que, para qualquer variável aleatória normal X com média positiva e coeficiente de variação $\delta_X \leq 1$, existe outra variável aleatória normal independente Y, com média positiva e com um coeficiente de variação que satisfaz algumas condições, tal que a razão $Z = X/Y$ pode ser bem próxima, dentro de um dado intervalo, de uma distribuição normal.

As condições estabelecidas por Qiao et al. (2006) e Díaz-Francés e Rubio (2013) devem ser checadas na estimação do *speedup*. Essas condições asseguram que a razão entre duas variáveis aleatórias normais independentes pode ser usada com segurança para estimar a razão entre duas médias

6.6.3 Média *Trimmed Mean*

Medida de tendência central é um valor único que tenta descrever as características de um conjunto de dados, identificando uma posição central dentro desse conjunto. A média é, dentre as medidas de tendência central, a mais usada e a mais confiável, pois usa todas as observações para ser computada e faz o melhor uso da amosta. Outras medidas de tendência central, como a moda e a mediana, utilizam apenas um ou poucos dados. A mediana usa apenas uma observação: aquela que divide a distribuição dos dados no meio. Segundo Jain (1991), a média atribui peso igual a cada observação, e por isso é sensível às observações que são muito maiores ou muito menores do que

as restantes, os chamados pontos discrepantes (em inglês, *outliers*). Os resultados dos experimentos realizados e apresentados no Capítulo 7 mostram a ocorrência de vários pontos discrepantes nas amostras coletadas.

Segundo Rao (1999) *apud* (Memória, 2004), na segunda metade do século passado houve uma mudança na pesquisa estatística para a utilização de métodos não-paramétricos, aplicáveis em amostras oriundas de qualquer distribuição, e de métodos paramétricos robustos, aqueles que não são influenciados por valores discrepantes. Embora de rápida implementação, os métodos não-paramétricos baseados em estatísticas de posto (*rank*) não tinham a eficiência dos métodos paramétricos.

Erceg-Hurn e Mirosevich (2008) referem-se aos métodos não-paramétricos, desenvolvidos antes da década de 1960, como métodos não-paramétricos clássicos e os consideram limitados em relação aos métodos robustos modernos (Wilcox e Keselman, 2003). Os autores argumentam que pesquisadores têm uma ideia equivocadas em relação aos métodos modernos, como a de que esses métodos envolvem o descarte de informação importante. Não faz sentido um teste ser mais preciso com menos informação, e essa é a razão da desconfiança em relação aos métodos modernos, como a média *trimmed mean*. Wilcox e Keselman (2003) enfatizam que a média *trimmed mean* de uma amostra não representa uma estimativa da média populacional, mas a média *trimmed mean* da população.

A média *trimmed mean* é um estimador robusto de tendência central da amostra. Para a média *trimmed mean*, $x\%$ das menores e das maiores observações são deletadas, e a média é calculada com as observações remanescentes. A mediana é, de fato, uma forma extrema da média *trimmed mean*, com a porcentagem de *trim* igual a 50%.

Exemplo 6.1 [Média *trimmed mean* (Erceg-Hurn e Mirosevich, 2008)] Considere o conjunto de dados:

$$1, 1, 1, 2, 2, 5, 5, 5, 6, 20, 40.$$

A média dos valores é 8, porém a maioria dos valores é menor ou igual a 6. A média não reflete de forma precisa a tendência central dos valores do conjunto, ela foi distorcida pelos valores discrepantes 20 e 40. A mediana é 5, mas a mediana descarta muita informação, toda informação menos o valor situado na posição central. Um bom balanço

entre a mediana e a média é a média *trimmed mean* de 20% (Wilcox e Keselman, 2003).

Ao remover os 20% dos maiores e os 20% dos menores valores, o conjunto de dados restantes fica da seguinte forma:

$$1, 2, 2, 5, 5, 5, 6.$$

A média *trimmed mean* é 3,71, que reflete os pontos centrais do conjunto de dados original de forma mais precisa do que a média com valor 8. □

A média *trimmed mean* é uma alternativa à média e mediana porque ela lida com os pontos discrepantes sem descartar a maioria da informação. Quando a distribuição dos dados é normal, a média e média *trimmed mean* coincidem.

6.7 Fatores a Explorar

Segundo Barr et al. (1995), existem três categorias de fatores que afetam o desempenho dos algoritmos em um experimento computacional: fatores do problema, fatores do algoritmo e fatores do ambiente de teste.

Cada categoria contém uma variedade de influências nos resultados dos testes e o experimentador deve ser sensato ao selecioná-los entre os que serão estudados, os que serão fixados em um determinado valor e os que não serão estudados. Os últimos são os fatores que podem ter efeito no desempenho do algoritmo, mas que não serão controlados, como a habilidade do programador, os processos trabalhando em segundo plano, o comportamento aleatório do algoritmo. O planejamento do experimento, de acordo com os objetivos definidos, determina como cada um desses fatores deve ser tratado no experimento.

A escolha dos fatores é de suma importância para os experimentos comparativos e descritivos, e deve ser cuidadosamente documentada.

- **Fatores do Problema.**

  Uma variedade de características do problema, como as suas dimensões e sua estrutura, podem afetar os resultados de uma dada observação. Esses efeitos, se houver, são importantes para avaliar a robustez do código. É importante testar os limites do código com instâncias de maior tamanho possível. Muitos efeitos

não aparecem em instâncias pequenas, e utilizar apenas instâncias de tamanho pequeno pode não produzir predições corretas para problemas maiores e mais realistas.

- **Fatores do Algoritmo.**
  A aplicação de uma metaheurística envolve a escolha de estratégias e parâmetros específicos. O processo de escolha deve ser bem documentado, uma vez que o desempenho de um algoritmo está intimamente ligado a essas estratégias e aos valores apropriados dos parâmetros.

  Um fator do algoritmo que deve ser abordado é o critério de parada. Metaheurísticas podem ser executadas por longos períodos de tempo, e o critério de parada é uma condição que faz a metaheurística parar. Não existe um critério de parada padrão, e a sua escolha deve ser criteriosa e bem documentada. Os critérios, em geral, são baseados em esforço computacional pré-definido ou na qualidade da solução.

  Para Johnson (2002), o tempo de execução não deve ser utilizado como critério de parada. Esse é um critério que não pode ser reproduzido.

- **Fatores do Ambiente de Teste.**
  São os fatores relacionados com a plataforma experimental, como o tipo do processador, tamanho da memória, frequência do *clock*, sistema operacional, linguagem de programação, compilador, programas executados em segundo plano, habilidades do programador.

  Numa situação ideal, ao comparar dois algoritmos, os fatores do ambiente de teste seriam idênticos. Porém, isso nem sempre ocorre, e o experimentador deve identificá-los e tentar isolar a sua influência nos resultados.

6.8 Planejamento

O planejamento experimental refere-se ao plano que o experimentador utiliza para coletar os dados. Um bom planejamento experimental é imparcial, alcança os objetivos do experimento, demonstra claramente o desempenho dos processos testados, descobre razões para o desempenho, permite uma análise racional que gera conclusões fundamentadas e reproduzíveis. A melhor maneira de atingir esses resultados é através do

uso de métodos estatísticos reconhecidos de planejamento experimental (Montgomery, 2009).

O planejamento de experimentos com base na estatística –em inglês, *statistical design of experiments (DOE)*– é um processo para coletar e analisar dados com um modelo estatístico, como o fatorial completo ou o quadrado latino, que permite obter conclusões válidas e objetivas. O planejamento do experimento e a análise estatística estão inter-relacionados.

Para Barr et al. (1995), quando os resultados do experimento variam com as configurações do teste (instâncias de problemas, configuração dos parâmetros, plataforma computacional), a metodologia estatística é a única abordagem objetiva para análise desses resultados.

Nesta etapa é importante definir os níveis de significância estatística e prática que serão considerados na análise do experimento. Rardin e Uzsoy (2001) ressaltam a importância de uma questão sobre a interpretação da significância estatística (determina se um efeito observado não é devido a um erro amostral), e da significância prática (determina se um efeito observado é grande o suficiente para ser considerado). O menor dos efeitos pode ser estatisticamente significante se foi coletada uma quantidade grande de amostras, tornando o erro aleatório muito pequeno. Os experimentos com metaheurísticas, que em geral são métodos não-determinísticos, demandam repetições de execuções do algoritmo para garantir a aplicabilidade de alguns testes estatísticos, como a análise de variância. Porém, o aumento de repetições pode tornar efeitos muito pequenos com significância estatística. A significância prática de um efeito não deve ser considerada se ele não for estatisticamente significante, mas a significância estatística de um efeito não prova a sua significância prática.

Se os parâmetros da população são conhecidos, pode-se estimar *a priori* o número de réplicas necessárias para que os testes estatísticos detectem efeitos com a significância e a potência desejadas. Caso contrário, a potência do teste pode ser calculada *post hoc* e, neste caso, corresponde à medida da sensibilidade do teste estatístico em detectar efeitos com a significância desejada (Montgomery e Runger, 2009).

No Capítulo 4, o planejamento e análise de experimentos foram descritos com maiores detalhes.

6.9 Execução

A execução é a etapa em que os dados são coletados através da execução do algoritmo em estudo. O experimentador deve garantir que o planejamento do experimento seja seguido com o cuidado de garantir a aleatoriedade – executar os testes na ordem aleatória definida – e, a uniformidade da plataforma computacional – manter o mais uniforme possível para que não cause efeitos nas medidas de desempenho.

Para McGeoch (1996), a experimentação é um processo em desenvolvimento e em evolução. A medida que novas descobertas surgem, novos questionamentos aparecem e, consequentemente, o experimento tem que ser ajustado. Esse processo evolutivo, alternando hipóteses e experimentos sequencialmente, é crucial para o sucesso no estudo experimental de algoritmos.

6.10 Análise

A análise é a etapa do experimento em que os dados coletados são transformados em informação. Neto et al. (2010) ressaltam a importância de se planejar os experimentos para que a análise dos dados coletados possa extrair informações conclusivas, através da transcrição de uma advertência feita por Sir R. A. Fisher: "Chamar o especialista em estatística depois que o experimento foi feito pode ser o mesmo que pedir a ele para fazer um exame *post-mortem*. Talvez ele consiga dizer de que foi que o experimento morreu".

O uso de métodos estatísticos na análise dos dados é fortemente recomendado pela maioria dos trabalhos sobre análise experimental de algoritmos, como Barr et al. (1995); Bartz-Beielstein (2006); Bartz-Beielstein e Preuss (2010); Coffin e Saltzman (2000); Hooker (1994, 1995); Johnson (2002); McGeoch (1996); Rardin e Uzsoy (2001), dentre outros.

Na Seção 4.4 estão descritos os procedimentos de análise estatística de um planejamento fatorial 2^k com maiores detalhes.

6.11 Apresentação dos Resultados

Nas seções anteriores foram levantadas várias questões importantes relacionadas à análise experimental que indicam que realizar um bom experimento para análise de algoritmos não é uma tarefa trivial. Entretanto, depois de planejar, conduzir o

experimento e analisar os dados, a documentação da experiência é outro desafio, pois apenas expor os resultados não é de muita utilidade. Para Bartz-Beielstein e Preuss (2010), a descrição do experimento deve garantir os seguintes pontos:

- Reprodução: o experimento descrito deve poder ser repetido, pelo menos com alto grau de similaridade.
- Resultados: mostrar os resultados, se eles correspondem às expectativas e se os resultados mostram significância estatística e prática.
- Interpretação: expor o que os resultados apontam em relação aos algoritmos e aos problemas. Identificar se é possível generalizar as conclusões.

O relatório do experimento, quando bem estruturado, pode fornecer ao leitor uma orientação padrão, com a descrição dos detalhes necessários para entender e, se for o caso, reproduzir o experimento. Para tal, Bartz-Beielstein e Preuss (2010) propõem a organização da apresentação em sete partes, mostradas a seguir:

1. Questões a investigar
   Descreve o objetivo geral da pesquisa.
2. Planejamento pré-experimental
   Resume as primeiras, possivelmente exploratívas, execuções do programa que vão direcionar para tarefa e configuração – partes 3 e 4, respectivamente. A escolha das instâncias de problemas e medidas de desempenho podem ser tomadas de acordo com os dados coletados nas execuções preliminares. O relato do planejamento pré-experimental deve incluir também resultados negativos, como por exemplo, modificações realizadas no algoritmo que não funcionaram ou uma instância que se mostrou muito difícil, caso esses resultados forneçam novas perspectivas.
3. Tarefa
   Concretiza as questões com os objetivos do experimento e estabelece afirmações científicas e suas hipóteses a serem testadas.
4. Configuração
   Especifica as instâncias de problemas, o algoritmo e seus parâmetros, as medidas

de desempenho, o ambiente de teste. As informações fornecidas nesta parte devem ser suficientes para se repetir o experimento.

5. Resultados e visualização

   Apresenta os dados brutos (ou filtrados) dos resultados experimentais e, adicionalmente, mostra visualizações básicas quando significativas.

6. Observações

   Descreve exceções ou padrões incomuns observados, sem avaliação subjetiva ou explicação. Por exemplo, pode valer a pena considerar as interações entre parâmetros.

7. Discussão

   Conclui sobre as hipóteses especificadas na parte 3 – Tarefa – e fornece interpretações subjetivas necessárias sobre as observações registradas. Tenta responder a questão: O que aprendemos?

A importância de se garantir resultados reprodutíveis foi realçada no incidente conhecido com *the Duke saga*[4]. Baggerly e Coombes (2009) causaram grande impacto no meio acadêmico relacionado à clínica médica, ao tentar reproduzir resultados apresentado em artigo científico publicado na revista Nature Medicine em 2006. Pesquisadores da universidade de Duke apresentaram uma técnica que possibilitava o tratamento de câncer personalizado, baseado no estudo do genoma. A não reprodutibilidade dos resultados apresentados possibilitou que um trabalho contendo erros e conclusões inválidas fosse aceito em revistas conceituadas, e testes com seres humanos iniciados. A partir desse incidente, o Instituto de Medicina americano (IOM) investigou o caso e apresentou um relatório (Micheel et al., 2012) com recomendações para fortalecer o desenvolvimento e avaliação de testes. Uma das recomendações é garantir a reprodutibilidade dos resultados através da documentação rigorosa e disponibilização dos dados, códigos, e outras informações necessárias para se reproduzir os resultados. Essa recomendação pode ser alcançada com o auxílio de ferramentas como *Sweave* (Leisch, 2002), programação anotada combinando LaTeX e R (R Core Team, 2012) e *knitr* (Xie, 2013a,b), pacote projetado para geração dinâmica de relatórios integrados com R, permitem que o código seja re-executado e os mesmos resultados alcançados.

[4]http://www.economist.com/node/21528593

6.12 Método de Avaliação dos AEP com Fatorial 2^k

A avaliação da influência de fatores no desempenho dos AEP através da aplicação do planejamento fatorial 2^k passa pelas seguintes etapas:

1. Definir os objetivos.

   Experimentos com AEP são em geral realizados para caracterizar ou descrever o desempenho do AEP através da estimação dos efeitos dos fatores que influenciam o seu desempenho (ver Seção 6.4).

2. Escolher medida de desempenho.

   Os AEP possuem as características dos algoritmos paralelos e dos algoritmos evolutivos. A medida de desempenho (ver Seção 6.5) mais comum na avaliação de algoritmos paralelos determinísticos é o *speedup*. Para algoritmos estocásticos como os AE, o cálculo do *speedup* corresponde a razão de dois valores estimados pela média (Equação 6.1). Os cuidados no cálculo e análise dos *speedups* foram detalhados na Seção 6.6.

3. Escolher fatores e níveis.

   Um experimento com algoritmos envolve um conjunto de variáveis dependentes chamadas de medidas de desempenho, que são afetadas por um conjunto de variáveis independentes chamadas de fatores. Como os objetivos do experimento são atingidos através da análise de observações desses fatores e níveis, eles devem ser escolhidos com esse propósito em mente (ver Seção 6.7).

4. Definir planejamento.

   O planejamento fatorial 2^k corresponde à combinação dos fatores e níveis definidos na etapa 3, que produz uma matriz do planejamento. Nessa matriz, cada linha corresponde a uma configuração de execução do AEP. Réplicas do experimento são obtidas ao se repetir as execuções para todas as linhas da matriz.

   Os níveis de significância serão usados na fase de análise dos dados (etapa 6) e devem ser definidos *a priori*. O nível de significância estatística corresponde à probabilidade tolerável de se aceitar ou rejeitar a hipótese nula em um teste estatístico. Um resultado é dito ter significância estatística se for improvável que

tenha ocorrido por acaso. A significância prática representa a diferença em comparações que é considerada de interesse do experimento. No caso do planejamento fatorial 2^k, a significância prática corresponde ao tamanho mínimo do efeito dos fatores e interações na resposta do experimento, para que estes sejam considerados significativos. Como mencionado anteriormente, a significância prática de um efeito não deve ser considerada se ele não for estatisticamente significante, mas a significância estatística de um efeito não prova a sua significância prática.

O cálculo do *speedup* precisa do tempo de execução sequencial e do tempo de execução paralela do algoritmo. É importante garantir as mesmas condições de carga de trabalho em ambos os algoritmos e, para isso, deve ser tomado o cuidado de se verificar entre os fatores explorados, quais podem influenciar o tempo de execução sequencial. Se o tamanho da população é um dos fatores estudados, ele deve ter o mesmo nível, ou valor, nas execuções sequenciais e nas execuções paralelas do AEP, cujos tempos serão utilizados na estimativa de um determinado *speedup*.

5. Executar o experimento.

   Um dos princípios básicos do planejamento experimental é a aleatorização. A ordem de execução das corridas experimentais deve ser aleatória. O PRNG pode auxiliar na definição dessa ordem. Desta forma, o efeito de fatores não-controláveis é minimizado.

6. Analisar dados coletados.

   Existem vários softwares estatísticos que são úteis para se montar e analisar um fatorial 2^k, como o software de código aberto R (R Core Team, 2012). Na Seção 4.4 estão descritos alguns dos procedimentos estatísticos utilizados de análise de um planejamento fatorial 2^k. A análise consiste em:

   (a) Estimar os *speedups* a partir dos tempos de execução sequenciais e paralelos do AEP .

      i. Analisar os tempos de execução sequenciais.
      Verificar a distribuição dos dados, presença de dados discrepantes e os coeficientes de variação.

ii. Analisar os tempos de execução paralela.

Verificar se os coeficientes de variação atendem às condições descritas na Seção 6.6.2 para que a distribuição do *speedup* se aproxime da distribuição normal.

Caso as condições não sejam satisfeitas aumentar a amostra e/ou aplicar a média *trimmed mean* (ver Seção 6.6.3).

iii. Calcular o *speedup* como uma razão ponderada (Equação 6.10) dos tempos de execução sequencial e dos tempos de execução paralela, para cada uma das 2^k corridas experimentais.

(b) Estimar os efeitos dos fatores e sua significância.

Dentre os métodos de estimação dos efeitos, o método pelo modelo de regressão linear (Seção 4.4.1) é fácil de se aplicar, principalmente quando se utiliza um software estatístico, como o R.

(c) Refinar o modelo com a remoção dos fatores e interações cujos efeitos não são significativos.

Existe uma variedade de funções disponíveis em softwares estatísticos que auxiliam nessa tarefa, como a função `stepAIC` do pacote `MASS` (Venables e Ripley, 2002) do R.

(d) Verificar a modelo.

A análise residual é executada para verificar se o modelo é adequado e se as pressuposições são válidas. A análise residual consiste na geração de gráficos dos resíduos e na execução de testes estatísticos formais. Se o modelo não é adequado ou se as pressuposições são violadas, é necessário verificar se uma transformação dos dados melhora a análise dos resíduos. Caso contrário, é necessário refinar o modelo (passo anterior).

(e) Interpretar os resultados.

Caso tenha sido necessário aplicar uma transformação dos dados, antes de interpretar os resultados é preciso fazer a transformação reversa. No caso da transformação logarítmica, uma transformação muito utilizada, é recomendado a escolha de uma base que seja mais fácil de ser interpretada, como a base 10. Para a transformação $y^* = log_{10}(y)$, a transformação reversa é dada por $\widehat{y} = 10^{\widehat{y^*}} = 10^{(\hat{\beta}_0+\hat{\beta}_1 x_1+\hat{\beta}_2 x_2 \cdots)} = 10^{\hat{\beta}_0} \times 10^{\hat{\beta}_1 x_1} \times 10^{\hat{\beta}_2 x_2} \ldots$ e, nesse caso, o

efeito de um fator pode ser interpretado como uma variação percentual da resposta.

A magnitude e os sinais dos efeitos dos fatores podem determinar quais fatores são importantes. O efeito principal de um fator deve ser interpretado individualmente, somente se não há evidência de que esse fator interage com outros fatores. Se a interação está presente, os fatores que interagem devem ser considerados conjuntamente. O sinal negativo de um efeito indica que uma mudança do nível − para o nível + reduzirá a resposta.

7. Relato dos resultados.

   Na Seção 6.11 foram descritos os cuidados necessários na etapa de publicação dos resultados e, principalmente, a importância de se garantir a reprodutibilidade dos experimentos. Assim como na etapa 6, de análise dos dados, nessa etapa é importante o uso de ferramentas computacionais como o pacote `knitr` do R (Xie, 2013a,b) para permitir que a análise possa ser re-executada e os resultados que foram relatados possam ser comprovados.

6.13 Considerações do Capítulo

Neste Capítulo foi apresentada uma metodologia para a experimentação com AE baseada em trabalhos relevantes que tratam da análise experimental de metaheurísticas e dos métodos utilizados na execução e relato de experimentos computacionais.

A constatação de que não há uma metodologia padrão nos relatos de experimentos com algoritmos é comum aos trabalhos citados. A falta dessa metodologia torna-se evidente em trabalhos com a descrição de um experimento computacional e o relato dos resultados obtidos, e é evidente a dificuldade de se reproduzir o experimento relatado ou de se extrapolar os resultados apresentados. A partir do levantamento realizado foram descritas as principais etapas para a condução de experimentos computacionais com AEP.

Para finalizar este capítulo, é válido referenciar Birattari (2004), que traz de maneira ilustrativa a comparação entre dois algoritmos de metaheurísticas Alg_A e Alg_B. Um experimento poderia buscar pela seguinte conclusão:

Proposição 6.1 *O algoritmo Alg_A tem melhor desempenho do que o algoritmo Alg_B.*

Ou vice-versa. De qualquer modo, uma conclusão tão geral e absoluta não pode ser obtida de um experimento computacional. Mais especificamente, a superioridade de um algoritmo sobre outro pode ser declarada somente depois de especificados os seguintes pontos: (1) sobre qual medida de desempenho e função de avaliação; e (2) sobre quais instâncias de problemas. Desta forma, a Proposição 6.1 ficaria melhor escrita da seguinte forma:

Proposição 6.2 *Sob dadas condições experimentais e sobre dadas instâncias, o algoritmo Alg_A tem melhor desempenho do que o algoritmo Alg_B.*

Onde condições experimentais referem-se à medida de desempenho e função de avaliação consideradas no experimento. A Proposição 6.2, apesar de menos genérica do que a Proposição 6.1, ainda apresenta uma falha. Como foi ressaltado por McGeoch (McGeoch, 1996), uma análise experimental de um algoritmo somente pode ser realizada com uma implementação do algoritmo – *programas de simulação* ou *programas de aplicação*. Está implícito, mas é relevante ressaltar que a implementação inclui não só o programa, mas também a configuração dos seus parâmetros. Metaheurísticas, em particular os AGP, possuem um número elevado de parâmetros.

Portanto, qualquer que seja o procedimento de ajuste de parâmetros usado nos algoritmos a serem comparados, a avaliação de desempenho e comparação somente é válida quando relacionada à configuração dos parâmetros do algoritmo. Para uma comparação justa, quando o procedimento de ajuste dos parâmetros usado não é mencionado, a formulação da proposição deve ser da seguinte forma:

Proposição 6.3 *Sob dadas condições experimentais e sobre dadas instâncias, o algoritmo Alg_A com a sua dada configuração tem melhor desempenho do que o algoritmo Alg_B com a sua configuração.*

E, para uma comparação mais justa ainda, isolando os efeitos do procedimento de ajuste de parâmetros sobre o desempenho, o mesmo procedimento deve ser usado em ambos os algoritmos. Assim, a proposição ficaria da seguinte maneira:

Proposição 6.4 *Sob dadas condições experimentais, sobre dadas instâncias, quando ambos os algoritmos são configurados usando o mesmo procedimento, Alg_A tem melhor*

desempenho do que o Alg_B.

Caso esse cuidado não seja tomado numa comparação entre dois algoritmos, o melhor desempenho de um deles pode ser resultado de um ajuste melhor dos seus parâmetros do que das qualidades do algoritmo.

Capítulo 7

Experimentos Realizados

Este capítulo relata o experimentos realizados e os resultados obtidos. Parte dos resultados relatados foram publicados em (Pais et al., 2011) e (Pais et al., 2014).

A utilização de um AGP em uma plataforma *multicore* implica em diversas escolhas a serem tomadas pelo desenvolvedor. Essas escolhas podem influenciar o desempenho dos AGPs. O primeiro estudo realizado tem como objetivo identificar quais fatores têm influencia no desempenho do AGP, quantificar essa influência e verificar a ocorrência de interação entre esses fatores.

7.1 Implementação do AGP-I

A implementação do Algoritmo Genético Paralelo, denominada AGP-I, segue o modelo de ilhas (ver Seção 2.3.1) que trabalha com múltiplas populações. A população é dividida entre os vários processos denominados de Subpopulações que trabalham em paralelo na evolução de sua população local. O tamanho da população permanece constante durante a execução. Um dos processos é chamado de Processo Central. O Processo Central, além de evoluir a sua população, sincroniza o início e o final do trabalho das outras Subpopulações. É também responsabilidade do Processo Central enviar às Subpopulações os parâmetros iniciais e indicar o início e fim do trabalho.

As Subpopulações trabalham em paralelo. Cada uma gera aleatoriamente a sua população local e inicia o ciclo de evolução. A cada intervalo de gerações previamente configurado (parâmetro *frequência de migração*) envia-se aos vizinhos, de acordo com a topologia de migração escolhida, um determinado número de indivíduos (parâmetro *taxa de migrantes*) selecionados aleatoriamente ou pelo valor da sua aptidão (parâme-

Algoritmo 7.1.1 Pseudo código Processo Central.

```
 1: Leitura dos parâmetros iniciais;
 2: Envio dos parâmetros para Subpopulações;
 3: enquanto (não ocorre a condição de parada) faça
 4:     Gera nova população aplicando operadores de cruzamento e mutação;
 5:     Avalia a população;
 6:     se (geração atual deve migrar) então
 7:         Seleciona indivíduos;
 8:         Envia indivíduos;
 9:         Recebe indivíduos;
10:         Substitui indivíduos locais pelos indivíduos recebidos;
11:         Avalia a população;
12:     fim se
13:     Recebe melhor indivíduo de cada Subpopulação;
14:     Verifica condição de parada;
15:     Avalie os resultados
16:     Selecione os sobreviventes para compor a nova geração
17: fim enquanto.
```

tro *seleção dos migrantes*). Em seguida, recebe os indivíduos que migram de outras Subpopulações vizinhas. O esquema de vizinhança é definido pelo parâmetro *topologia de migração* (Figura 7.1). Os indivíduos que chegam são posicionados na população local substituindo indivíduos escolhidos aleatoriamente ou os com menor aptidão (parâmetro *posicionamento dos migrantes*), com exceção do melhor indivíduo local que tem seu lugar garantido. O próximo passo é enviar ao Processo Central o seu melhor indivíduo e aguardar o retorno com o comando para continuar ou finalizar, como mostra o Algoritmo 7.1.1.

Algoritmo 7.1.2 Pseudo código Subpopulações.

```
 1: Recebe do Processo Central os parâmetros iniciais;
 2: Inicia a população local;
 3: enquanto (status == continuar) faça
 4:     Gera nova população aplicando operadores de cruzamento e mutação;
 5:     Avalia a população;
 6:     se (geração atual deve migrar) então
 7:         Seleciona indivíduos;
 8:         Envia indivíduos;
 9:         Recebe indivíduos;
10:         Substitui indivíduos locais pelos indivíduos recebidos;
11:         Avalia a população;
12:     fim se
13:     Envia melhor indivíduo para Processo Central;
14:     Recebe status do Processo Central;
15: fim enquanto.
```

Em ambos, Processo Central e Subpopulações, as linhas de código 4 – 11 mostradas nos Algoritmos 7.1.1 e 7.1.2 são as mesmas. A diferença entre os dois está no controle do sincronismo entre os processos executado pelo Processo Central. O algoritmo AGP-I foi implementado em C/C++, utilizando a biblioteca de troca de mensagens entre

processos MPICH2 (MPICH, 2011) (ver Seção 3.5.1).

A topologia de migração define como estão conectadas as Subpopulações e, desta forma, determina a vizinhança e direciona a migração. As topologias de migração investigadas são: anel e totalmente ligada. A topologia em anel (Figura 7.1 a) mostra que, por exemplo, a Subpopulação 1 envia para a Subpopulação 2 e recebe da Subpopulação 4. Na topologia totalmente ligada (Figura 7.1 b) a Subpopulação 1 envia para todas as Subpopulações e recebe de todas as Subpopulações.

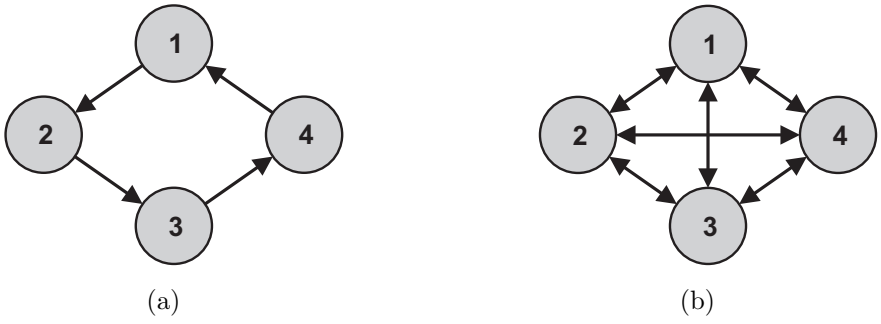


Figura 7.1: *Esquema das Topologias de Migração entre 4 Subpopulações: (a) em anel e (b) totalmente ligada.*

Os AGs são processos estocásticos que usam intensamente geradores de números aleatórios (Tomassini, 2005). Gerar números aleatórios não é uma tarefa fácil para um computador, uma vez que se trata de uma máquina determinística e é impossível criar números aleatórios verdadeiros sem algum hardware adicional.

O termo pseudo-aleatoriedade refere-se aos números aleatórios gerados por computador. O prefixo "pseudo" é usado para distinguir esse tipo de número de um "verdadeiro" número aleatório gerado por um processo físico , como por exemplo, deterioração radioativa. Geradores de números pseudo-aleatórios (PRNG - *Pseudo Random Numbers Generators*, em inglês) são algoritmos que, dado um valor inicial chamado de semente, geram uma sequência de números aparentemente aleatória. A ideia é que a sequência gerada de números pseudo-aleatórios se comportem como uma sequência de números verdadeiramente aleatórios, mas que são repetidos após um período fixo. É desejável que esse período seja o mais longo possível.

A diferença entre PRNGs de alta e baixa qualidade é que os de alta qualidade possuem uma melhor estrutura oculta mais difícil de ser detectada por testes estatísticos de aleatoriedade e geram menos interferência em aplicações específicas. Os mais utilizados

são: *Mersenne Twister* (MT), *The Mother-of-All Random Generators* (MoA), *SIMD-oriented Fast Mersenne Twister* (SFMT) e *Well Equidistributed Long-period Linear* (WELL).

Na implementação do AGP-I foi utilizado o PRNG SMFT1, uma combinação do SFMT (Saito e Matsumoto, 2008) com o MoA (Marsaglia, 1994), da biblioteca disponibilizada por Agner Fog via GNU *General Public License* (Fog, 2010).

7.2 Objetivo dos Experimentos

O objetivo dos experimentos realizados é a determinação dos fatores que influenciam o desempenho do AGP-I na resolução da minimização das funções Rastrigin e Rosenbrock em computadores com processador *multicore* Intel Xeon ou Intel i7. Busca-se descobrir como e quanto o aumento de desempenho do AGP-I na solução das duas funções em processador *multicore* é determinado pela sua configuração e como os parâmetros de configuração podem influenciar-se mutuamente.

7.3 Instâncias de Teste

Nos testes foram escolhidas duas funções multimodais muito utilizadas em *benchmarks* de otimização conhecidos, como o CEC 2010 (Táng et al., 2010): a função Rastrigin e a função Rosenbrock

A função Rastrigin é uma função contínua de otimização global, multimodal e separável (ver Definição 2.3.1). A função Rastrigin é dada pela Equação 7.1, descrita a seguir.

$$f_{Ras}(X) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)), \tag{7.1}$$

onde n é o número de dimensões, $-5.12 \leq x_i \leq 5.12$, mínimo global é $f_{Ras}(X^*) = 0$, e $X^* = [0, 0, ..., 0]$. Foram utilizadas 30 dimensões ($n = 30$) nos testes realizados neste trabalho.

A função Rosenbrock está presente no conjunto de funções testadas por De Jong, Czarn et al. (2004b), entre outros *benchmarks*. É uma função não separável[1], uma característica dos problemas difíceis de resolver, e é multimodal para dimensões maiores que dois (Táng et al., 2010). A função Rosenbrock é definida pela Equação 7.2.

[1]Uma função *não separável* $f(x)$ é chamada de *m-não separável* se pelo menos m dos seus parâmetros não são independentes (Táng et al., 2010).

$$f_{Ros}(\mathbf{x}) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right], \tag{7.2}$$

onde n é o número de dimensões, $-2.0 \leq x_i \leq 2.0$, e o mínimo global é $f_{Ros}(X^*) = 0$, e $X^* = [1, 1, ..., 1]$. Nos testes realizados são utilizadas 30 dimensões ($n = 30$) e o valor próximo da solução ótima utilizado na condição de parada é de 10^{-10}.

7.4 Resposta dos Experimentos

A resposta dos exerimentos foi o *speedup* do AGP-I. Para tal, a cada execução do AGP-I foi medido o tempo total de processamento (*wall clock*) em milissegundos. Com o tempo de execução de cada corrida experimental foi avaliada a melhora no desempenho do AGP-I através da medida *speedup* (Equação 6.1). O *speedup* é calculado pela divisão do tempo de execução do programa sequencial pelo tempo de execução do programa paralelo. Segundo Tomassini (2005), para se ter uma medida de desempenho mais justa e significativa, no caso dos algoritmos evolucionários paralelos, deve-se usar como tempo de execução sequencial aquele medido na execução do mesmo algoritmo paralelo configurado com apenas uma subpopulação. Segundo a classificação de Alba (2002), esse *speedup* é do tipo II.A.1, ou seja, *speedup* fraco, com melhor solução e ortodoxo (ver Seção 6.5.3).

Neste trabalho considerou-se o tempo sequencial para o cálculo do *speedup* como o tempo médio de execução do AGP-I configurado com apenas uma subpopulação. Para obter o tempo médio de execuções sequenciais, foram incluídos na matriz de planejamento experimentos com uma única ilha (um processo). Os tempos de execução paralelos foram coletados com a execução paralela do AGP-I na resolução das funções de teste descritas na Seção 7.3, com configurações definidas pelas corridas experimentais da matriz de planejamento do fatorial 2^k.

7.4.1 Significância Estatística e Significância Prática

Nas análises, foram considerados o nível de 5% de significância estatística e significância prática de efeitos maiores que 10% do *speedup* médio.

7.5 Plataforma Computacional

Para a execução dos testes foram utilizados:

- um computador com processador Intel® Xeon® E5504, com duas CPU com quatro núcleos cada, totalizando 8 núcleos a 2GHz, 64KB *cache* L1, 512KB *cache* L2, 4MB *cache* L3, FSB a 1333 MHz, 4 GB com sistema operacional Ubuntu 10.04 LTS (*kernel* 2.6.32); e

- um computador com processador processador Intel® CORE^{TM}I7® 2630QM, 2GHz, com uma CPU com quatro núcleos, *HyperThreading*, *Turbo Boost*, 32K *cache* L1, 256KB *cache* L2, 6MB *cache* L3 compartilhada, 6 GB, com sistema operacional Ubuntu 10.04 LTS (*kernel* 2.6.32).

7.6 Fatores e Níveis

Os fatores escolhidos correspondem a parâmetros de configuração do AGP-I relacionados à migração (ver Seção 2.3.1) e ao processador *multicore* usado. Os fatores e os seus níveis estão listados na Tabela 7.1.

Tabela 7.1: *Níveis dos fatores analisados pelo planejamento experimental.*

#	Fator	Descrição	Nível -	Nível +
1	*Proc*	Número de Subpopulações	4	16
2	*Top*	Topologia da migração	anel	totalmente ligada
3	*Rate*	Frequência da migração	10 gerações	100 gerações
4	*Pop*	População total	1600 indivíduos	3200 indivíduos
5	*Nindv*	Taxa de migrantes	1 indivíduo	5 indivíduos
6	*Sel*	Seleção dos migrantes	Aleatória	Melhor(es) individuo(s)
7	*Rep*	Posicionamento dos migrantes	Aleatória	Substituição dos piores indivíduos
8	*Hw*	Processador *multicore*	Xeon	i7

- Os níveis para o fator **Proc** (número de subpopulações) foram definidos em função do número de núcleos existentes no processador: metade e dobro do número de núcleos.

- O fator **Top** (topologia de migração) pode assumir o valor anel ou totalmente ligada.

- O fator **Rate** (frequência de migração) representa a frequência com que ocorrem as migrações: a cada 10 ou 100 gerações.

- O fator **Pop** (população total) corresponde ao tamanho do total da população do AGP-I que é divida entre as subpopulações.

- O fator **Nindv** (taxa de migrantes) representa o número de indivíduos que migram de uma subpopulação para outra.

- O fator **Sel** (forma de seleção dos migrantes) representa o método de seleção dos indivíduos locais que participarão da migração; aleatória ou escolha dos melhores.

- O fator **Rep** (tipo de posicionamento dos migrantes) determina de que forma os indivíduos que chegam a uma subpopulação são posicionados: de forma aleatória ou nas posições dos piores indivíduos.

- O fator **Hw** (processador *multicore*) representa duas arquiteturas de processadores diferentes: o Xeon com oito núcleos físicos e o i7 com quatro núcleos físicos e quatro virtuais.

Os fatores de configuração do AGP-I que foram utilizados nos testes e cujos valores não foram alterados nas execuções são mostrados na Tabela 7.2.

Tabela 7.2: *Valores dos fatores fixos.*

Fator	Valor
Seleção	Torneio (n = 2)
Taxa de cruzamento	0.9
Taxa de mutação	0.001
Operador de cruzamento	*Simulated Binary Crossover Operator* (SBX)
Condição de parada	solução ótima (função Rastrigin) e solução aproximada $\leq 10^{-10}$ (função Rosenbrock)
Tamanho do problema	30 dimensões

7.7 Planejamento e Execução

O planejamento adotado foi o planejamento fatorial completo 2^k totalmente aleatorizado. Além das corridas experimentais do fatorial, foram adicionadas corridas para a estimação do tempo sequencial.

Na estimação do *speedup* é necessário que a mesma carga de trabalho seja executada pela versão sequencial e a paralela (ver Seção 7.4). Essa carga de trabalho para o AGP-I corresponde ao número de indivíduos da população, que é um dos fatores analisados pelo planejamento fatorial 2^k, o fator Pop. Ao estimar o *speedup*, os tempos sequenciais e paralelos devem ser medidos no mesmo processador *multicore*, o fator Hw. Portanto, para garantir que o *speedup* reflita a comparação do desempenho sequencial e

paralelo resolvendo o mesmo problema e com carga de trabalho semelhantes, os tempos sequenciais foram capturados com o AGP-I sendo executado com uma ilha, tamanho da população variando entre 1600 e 3200 e processador *multicore* variando entre Xeon e i7. Esses valores correspondem aos níveis + e - do fator Pop e Hw da Tabela 7.1.

A matriz de planejamento do experimento fatorial 2^8 tem 256 linhas para as corridas experimentais que correspondem às execuções paralelas. Foram adicionadas quatro corridas experimentais para as execuções sequenciais, uma para cada combinação de processador e tamanho da população, totalizando 260 corridas experimentais.

A natureza probabilística dos AGs produz resultados diferentes para execuções repetidas com mesmas entradas, sendo necessárias várias réplicas das corridas com a ordem de execução definida aleatoriamente. O número inicial de réplicas foi igual a 40, e o PRNG foi utilizado para definir a ordem das execuções.

7.8 Notação Adotada

Ao longo do texto, os tempos de execução sequencial são denotados por X_{pmj}, na qual p é o tamanho da população, m o processador *multicore*, e j a réplica. Os tempos de execução paralelos são denotados por Y_{ij}, em que i é a corrida experimental e j a réplica. As médias e os desvios padrões amostrais foram usados para estimar os coeficientes de variação $\hat{\delta}$.

7.9 Função Rosenbrock

Nesta seção são apresentados os resultados e análises do experimento realizado para a função Rosenbrock.

7.9.1 Análise dos Tempos Sequenciais

Os tempos sequenciais foram obtidos através da execução do AGP-I com apenas uma subpopulação e variando-se o fator Pop e fator Hw em dois níveis. Os níveis do fator Pop foram de 1600 e 3200 indivíduos, e do fator Hw foi Xeon ou i7 (Tabela 7.1). A Tabela 7.3 apresenta um sumário dos tempos de execução sequencial obtidos com o tamanho $n = 40$ da amostra: a média $\overline{X}$, o desvio padrão S, a mediana, o valor mínimo e valor máximo, o coeficiente de variação $\hat{\delta}_X$ de X, o coeficiente de variação $\hat{\delta}_{\overline{X}}$ de $\overline{X}$ e o erro padrão SE.

Tabela 7.3: *Tempos de execução (ms) do sequencial AGP-I para a função Rosenbrock.*

	$\overline{X}$	S	mediana	min	max	$\hat{\delta}_X$	$\hat{\delta}_{\overline{X}}$	SE
Xeon 1600	219.152,9	1.384,5	219.202,4	216.867	223.377	0.01	0.00	218,9
Xeon 3200	431.861,3	1.996,0	431.824,1	426.218	435.554	0,00	0,00	315,6
i7 1600	163.230,2	26.669,6	149.646,0	145.760	214.439	0,16	0,03	4.216,8
i7 3200	310.087,6	50.716,7	284.045,5	280.131	414.952	0,16	0,03	8.019,0

A Figura 7.2 exibe dois histogramas dos tempos de execução sequencial do AGP-I para a função Rosenbrock. No topo da figura, sobre o histograma estão desenhadas as linhas que representam as estimativas da função densidade de probabilidade dos tempos sequenciais para Pop = 1600 e Pop = 3200. Na parte inferior, sobre o histograma estão desenhadas as linhas que representam as estimativas da função densidade de probabilidade dos tempos sequenciais para Hw = Xeon e Hw = i7.

Observando-se a Figura 7.2 da esquerda para a direita, é possível identificar quatro grupos distintos de dados: i7 com 1600, Xeon com 1600, i7 com 3200 e Xeon com 3200. As médias $\overline{X}$ e medianas na Tabela 7.3 confirmam que esses grupos de dados estão centrados nas posições 163×10^3, 219×10^3, 310×10^3 e 431×10^3 do eixo x do histograma.

As funções densidades, estimadas para a distribuição dos tempos sequenciais do AGP-I para a função Rosenbrock, são exibidas mais uma vez na Figura 7.3. Do lado esquerdo, encontra-se a comparação entre as funções densidade para populações com

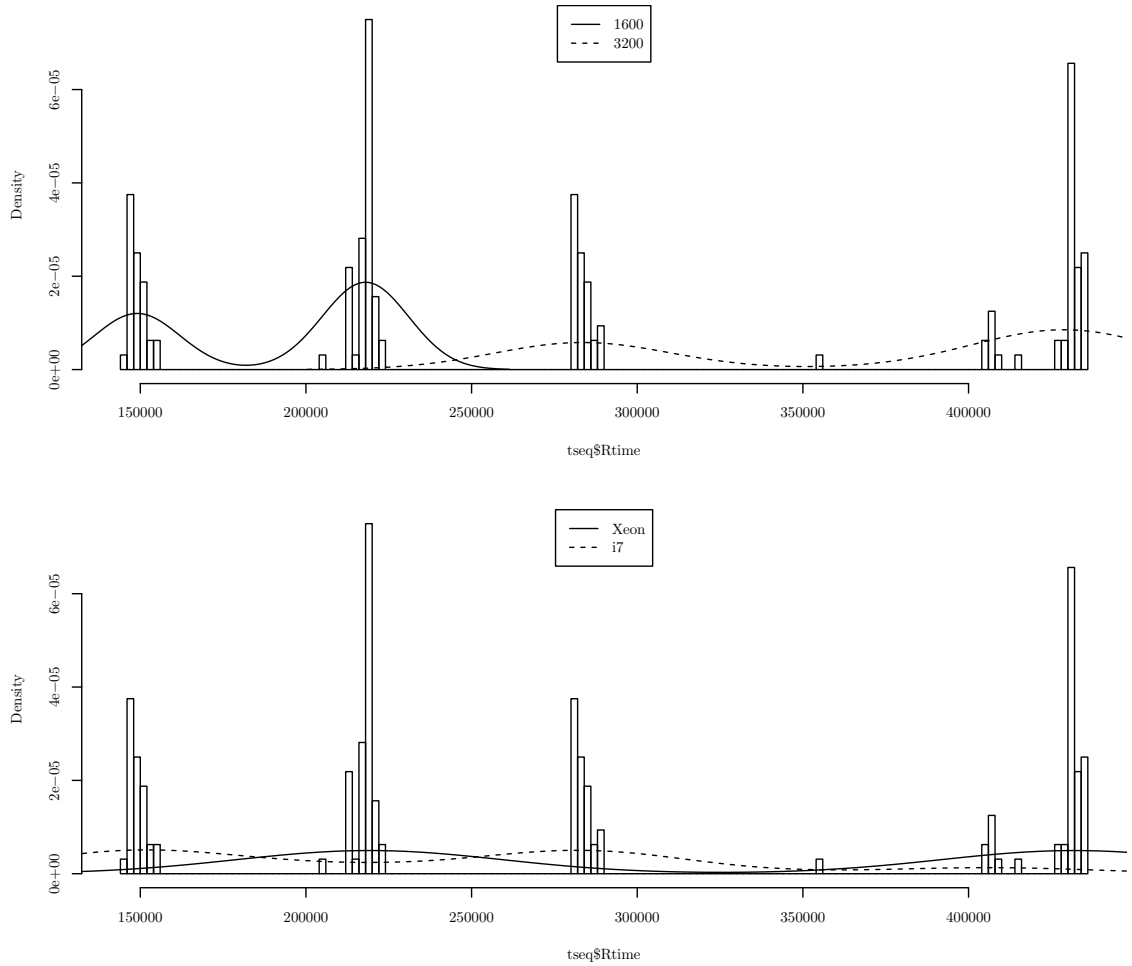


Figura 7.2: *Histogramas dos tempos de execução sequencial do AGP-I para a função Rosenbrock com linhas do contorno da função densidade estimada para os tempos agrupados por tamanho da população (histograma superior) e por processador multicore (histograma inferior).*

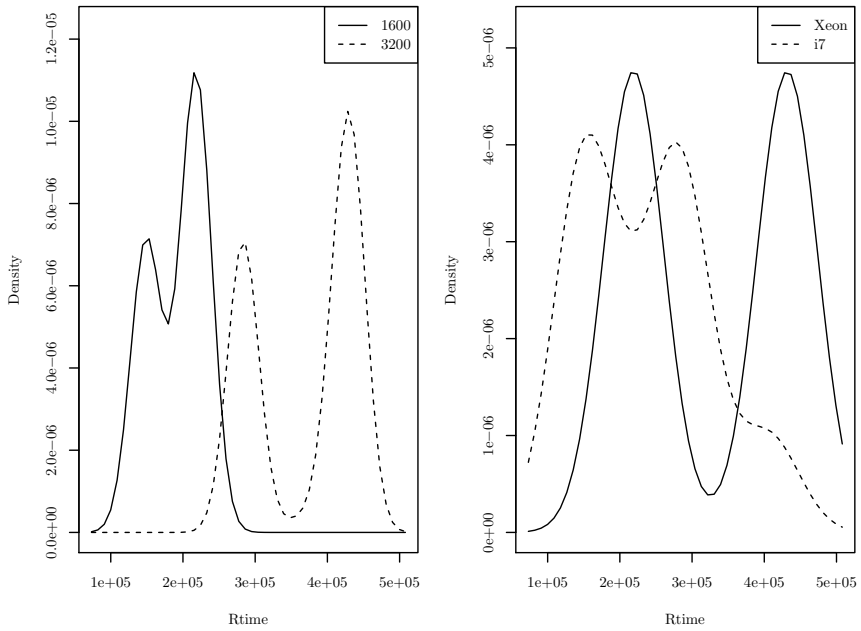


Figura 7.3: *Comparação das funções densidade estimadas para os grupos de execuções, do lado esquerdo da figura, separados pelo tamanho da população, e do lado direito pelo processadores multicore.*

1600 e 3200 indivíduos e, do lado direito, a comparação entre as funções densidade para o processador Xeon e o i7. O gráfico do lado esquerdo apresenta uma diferença entre os tempos sequenciais para populações distintas, onde os tempos são menores para populações com 1600 indivíduos.

Os diagramas de caixa (*boxplot*) dos tempos sequenciais são exibidos nas Figuras 7.4, 7.5 e 7.6. A Figura 7.4 exibe três gráficos com diagramas de caixa dos tempos sequenciais agrupados por: tamanho da população, processador *multicore* e ambos. O primeiro gráfico, à esquerda, apresenta os tempos sequenciais maiores para populações com 3200 do que as com 1600 indivíduos. O gráfico do meio mostra que a metade central dos tempos sequenciais no Xeon é superior aos tempos no processador i7. No terceiro gráfico é possível vislumbrar a presença de valores discrepantes nos valores obtidos para o processador i7.

As diferenças nos tempos sequenciais coletados no processador Xeon podem ser melhor visualizados nos diagramas de caixa da Figura 7.5. Os tempos sequenciais com população 1600 foram todos menores que os tempos para população com 3200.

Para os tempos sequenciais coletados no processador i7, as diferenças são exibidas através dos diagramas de caixa da Figura 7.6. No primeiro gráfico, à esquerda, 100% dos tempos sequenciais para populações com 1600 estão abaixo dos tempos com 3200

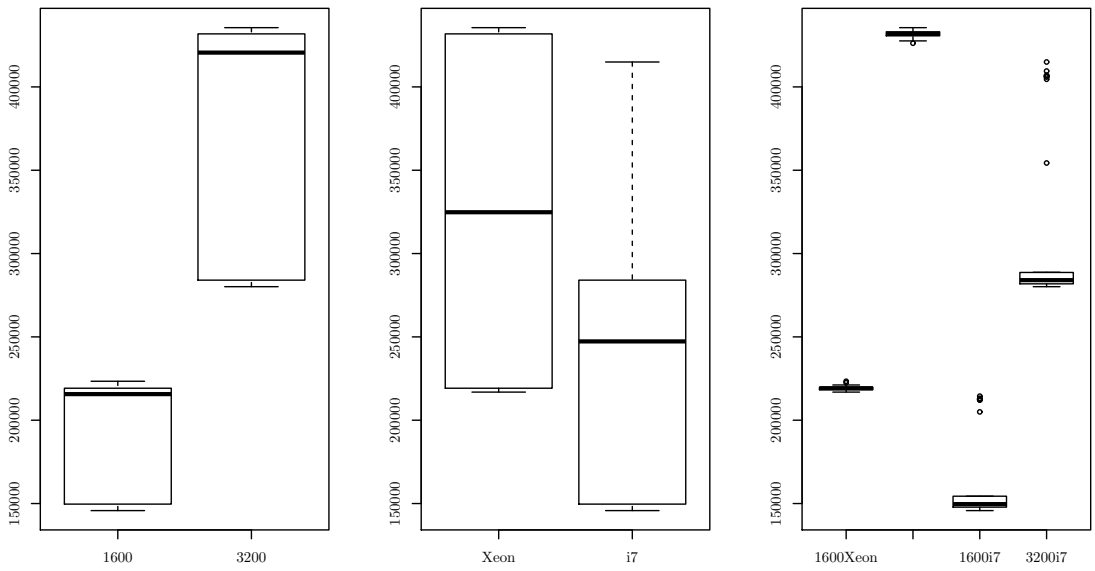


Figura 7.4: *Diagramas de caixa (boxplot) dos tempos de execução sequencial do AGP-I para a função Rosenbrock com tempos de execução agrupados por: tamanho de população, processador multicore.*

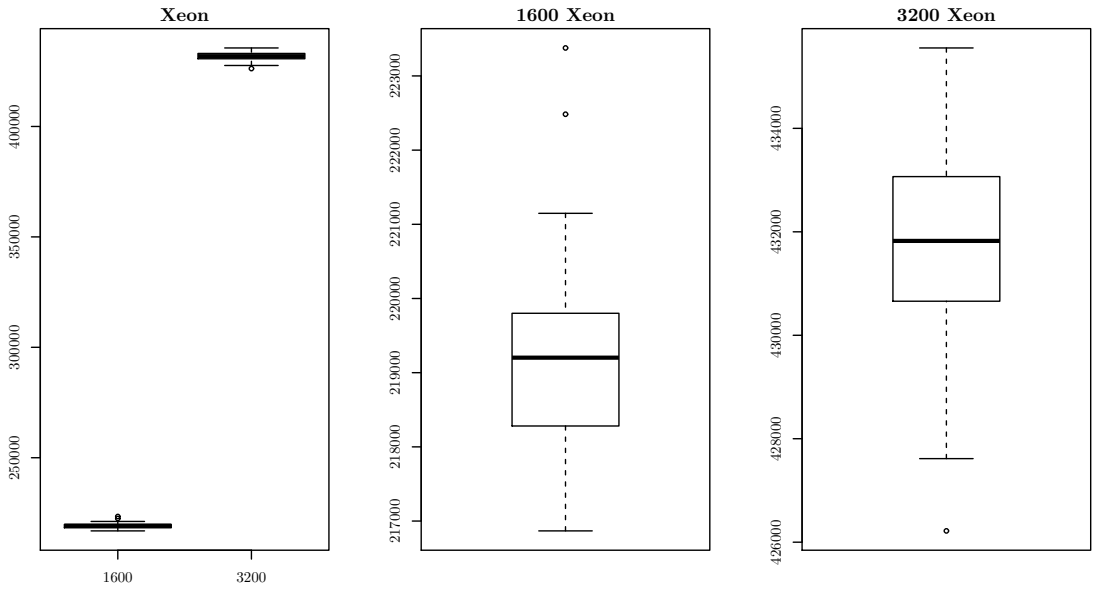


Figura 7.5: *Diagramas de caixa (boxplot) dos tempos de execução sequencial do AGP-I para a função Rosenbrock no Xeon.*

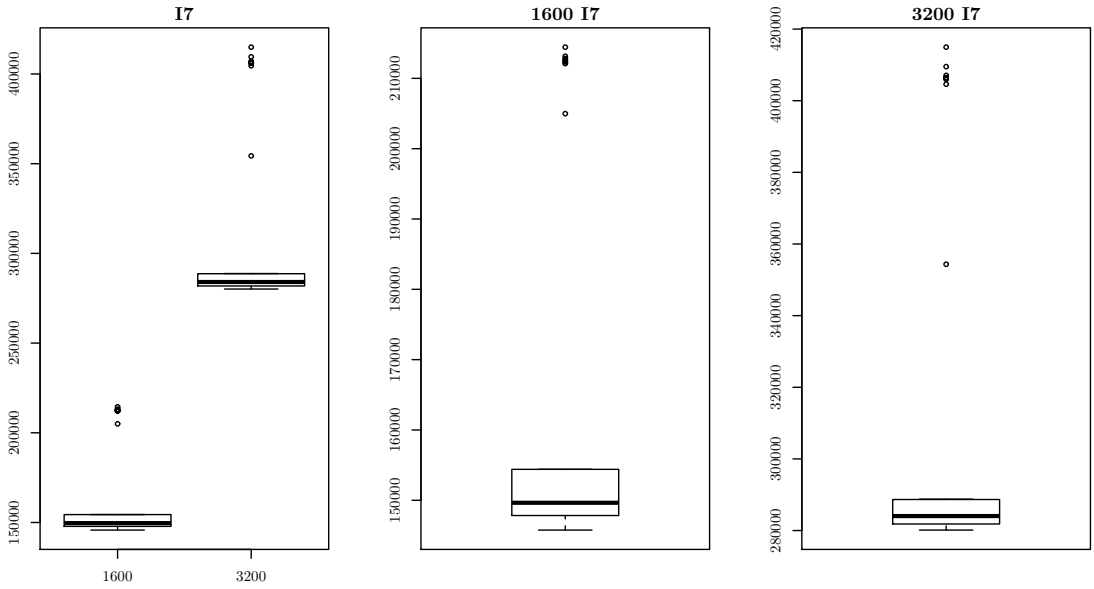


Figura 7.6: *Diagramas de caixa (boxplot) dos tempos de execução sequencial do AGP-I para a função Rosenbrock no i7.*

indivíduos. Os diagramas de caixa para ambos tamanhos de população apresentaram pontos discrepantes.

Os tempos sequenciais foram separados nos grupos: Xeon com 1600, Xeon com 3200, i7 com 1600 e i7 com 3200. Para cada um dos grupos foram gerados gráficos exploratórios apresentados nas Figuras 7.7, 7.8, 7.9 e 7.10, respectivamente. Essas figuras apresentam, no topo, da esquerda para direita, o diagrama de caixa e o histograma com a linha da função densidade estimada para os tempos sequenciais. Em baixo, da esquerda para a direita, o diagrama de caixa para as quantidades de gerações, e o histograma das quantidades de geração. A quantidade de gerações está relacionada à carga de trabalho executada pelo AGP-I e é determinada pela semente do PRNG. A distribuição dos tempos sequenciais deveria ser próxima da distribuição da quantidade de gerações. Para execuções sequenciais no processador Xeon essa proximidade pode ser visualizada nas Figuras 7.7 e 7.8. A distribuição dos tempos sequenciais no processador i7 não apresentaram proximidade com a distribuição da quantidade de gerações, como é exibido nas Figuras 7.9 e 7.10.

A quantidade de gerações e o tempo de execução sequencial deveriam apresentar correlação positiva, uma vez que ambos crescem com a carga de trabalho do AGP-I. A Tabela 7.4 exibe estimativas dos coeficientes de correlação ρ de Pearson e o valor-p do teste da hipótese do coeficiente ser zero. A correlação para Xeon foi positiva e significativa, enquanto que para o i7 não foi possível descartar a hipótese de que os

coeficientes foram zero.

Tabela 7.4: *Estimativas dos coeficientes de correlação dos tempos de execução sequencial e a quantidade de gerações do AGP-I para função Rosenbrock.*

Grupo	ρ	valor-p
Xeon 1600	0,70	0,0000
Xeon 3200	0,50	0,0011
I7 1600	-0,11	0,4907
I7 3200	0,07	0,6559

As Figuras 7.7, 7.8, 7.9 e 7.10 confirmam as estimativas dos coeficientes de correlação. As Figuras 7.9 e 7.10 exibem pontos discrepantes para os tempos de execução e não apresentam pontos discrepantes para os gráficos das quantidades de gerações. Um dada sequência de números pseudo-aleatórios gerada pelo PRNG produz sempre a mesma quantidade de gerações para se chegar a uma mesma solução ótima. Podemos dizer que a quantidade de gerações necessárias para resolver o problema é determinada pela sequência de números pseudo-aleatórios gerada pelo PRNG. Se a distribuição dos tempos de execução não apresenta a mesma variabilidade, então outros fatores, além da semente do PRNG, afetaram a variabilidade dos tempos de execução no processador i7.

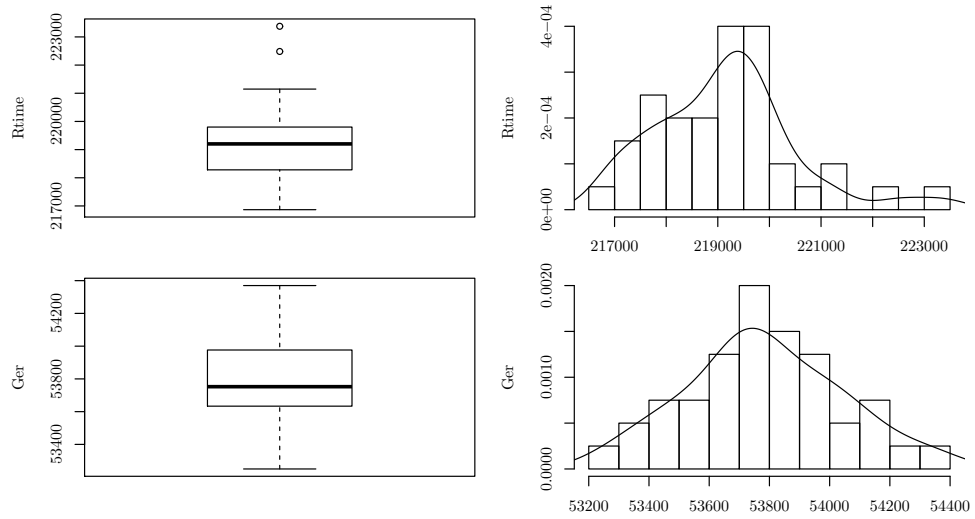


Figura 7.7: *Gráficos exploratórios para os tempos de execução sequencial e quantidades de gerações do AGP-I para a função Rosenbrock no Xeon com população de 1600.*

Os tempos de execução sequencial foram coletados com o objetivo de estimar o tempo sequencial que ocupa a posição do numerador X na fração X/Y da fórmula do *speedup* s_p (6.1). Segundo o Teorema 6.6.1, uma das condições para que a distribuição de X/Y se aproxime da distribuição normal é que o coeficiente de variação δ_X deve ser menor que um. Os coeficientes de variação de X para os grupos $Xeon1600$, $Xeon3200$, $i71600$ e $i73200$ foram menores que um. Para Qiao et al. (2006), apenas o coeficiente de

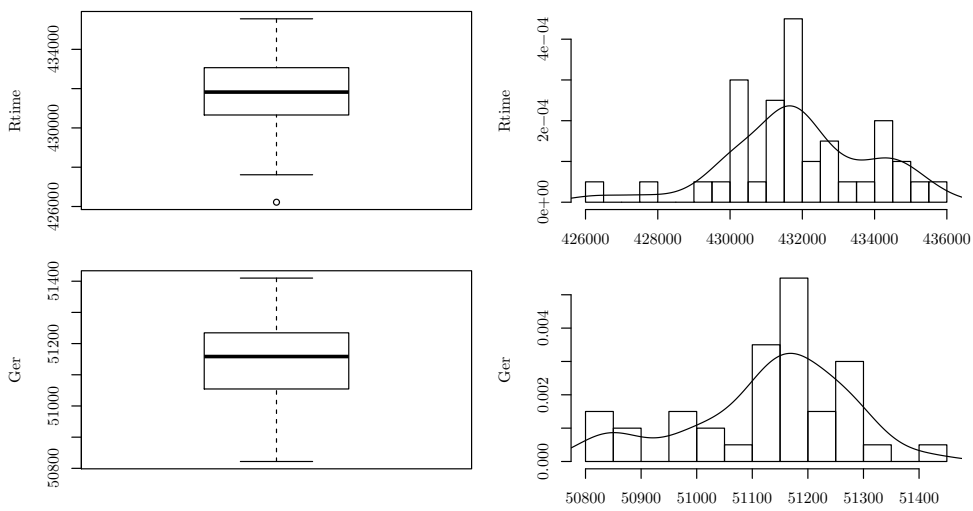


Figura 7.8: *Gráficos exploratórios para os tempos de execução sequencial e quantidades de gerações do AGP-I para a função Rosenbrock no Xeon com população de 3200.*

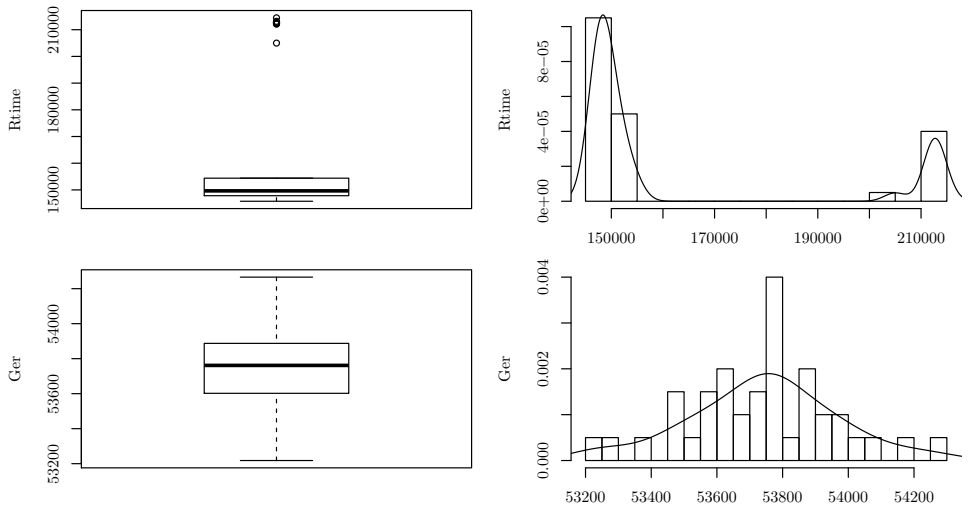


Figura 7.9: *Gráficos exploratórios para os tempos de execução sequencial e quantidades de gerações do AGP-I para a função Rosenbrock no i7 com população de 1600.*

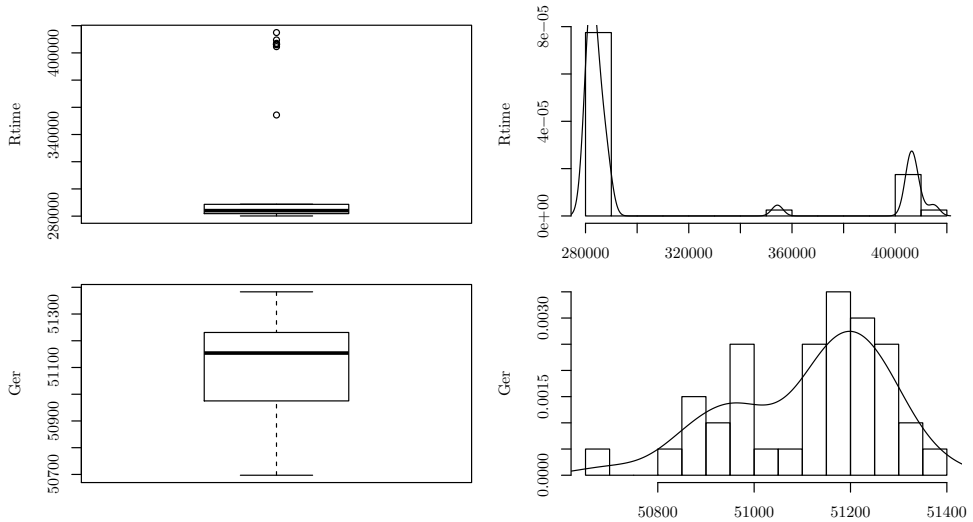


Figura 7.10: *Gráficos exploratórios para os tempos de execução sequencial e quantidades de gerações do AGP-I para a função Rosenbrock no i7 com população de 3200.*

variação do denominador Y é que importa. Portanto, de acordo com Qiao et al. (2006) e Díaz-Francés e Rubio (2013), as condições relativas ao numerador X do *speedup* foram satisfeitas. As condições relativas ao denominador Y foram verificas e estão relatadas na próxima seção (Seção 7.9.2).

Neste trabalho, a média foi adotada como medida de tendência central dos dados para o cálculo do *speedup*. A média é sensível às observações que são muito maiores ou muito menores do que as restantes, os chamados pontos discrepantes ou *outliers*.

As transformações dos dados, como a transformação logarítmica, podem ser utilizadas para amenizar a assimetria causada pela presença dos pontos discrepantes. A Figura 7.11 exibe os diagramas de caixa para os tempos de execução sequencial do AGP-I para função Rosenbrock no processador i7 transformados pela função logarítmica na base 10. Os pontos discrepantes continuaram presentes nos gráficos.

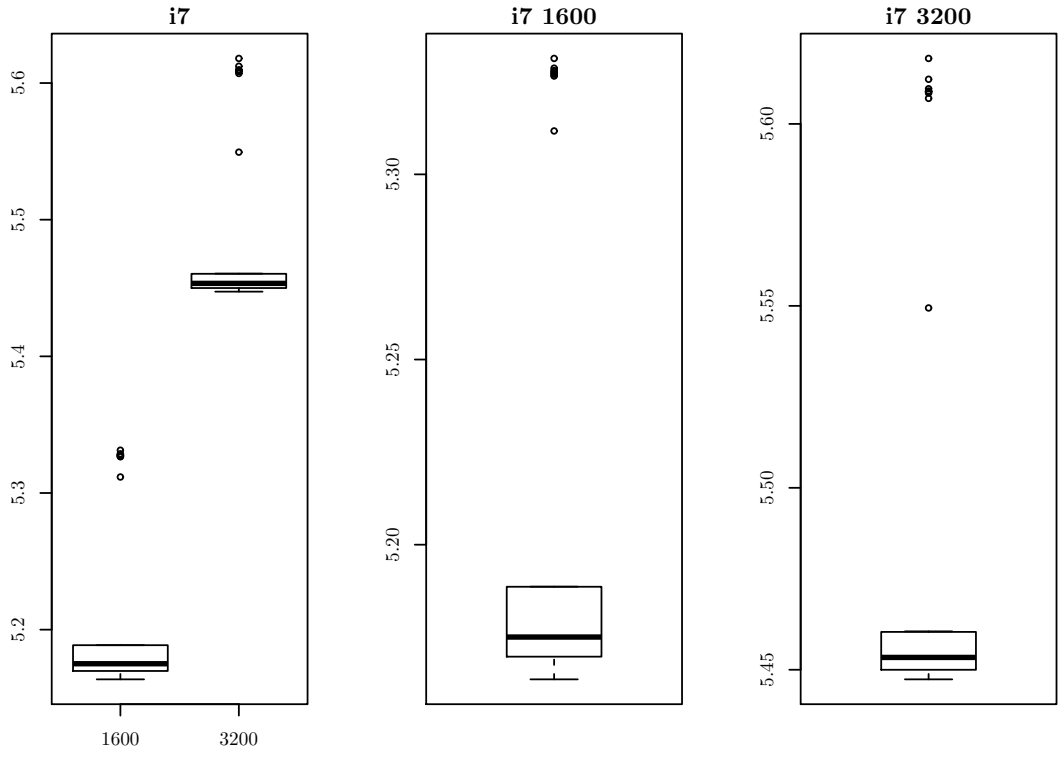


Figura 7.11: *Diagramas de caixa para os tempos de execução sequencial do AGP-I para função Rosenbrock no processador i7 após a transformação por* log_{10}.

É recomendado que os pontos discrepantes não sejam eliminados antes de estudados. Na análise dos tempos de execução sequencial do AGP-I, a quantidade de pontos discrepantes foi relativamente pequena e foi viável verificá-los um a um. Os tempos de execução com valores acima da maioria não foram consequência de um número de gerações discrepante, ou seja, a variabilidade dos tempos de execução discrepantes não

pode ser atribuída à semente do PRNG.

Quando a quantidade de pontos discrepantes é maior, a verificação de cada um desses pontos torna-se mais onerosa. Tem-se, então, a necessidade de métodos que não envolvam a investigação de cada um dos pontos, de métodos automáticos como o *trimmed mean* (Wilcox e Keselman, 2003).

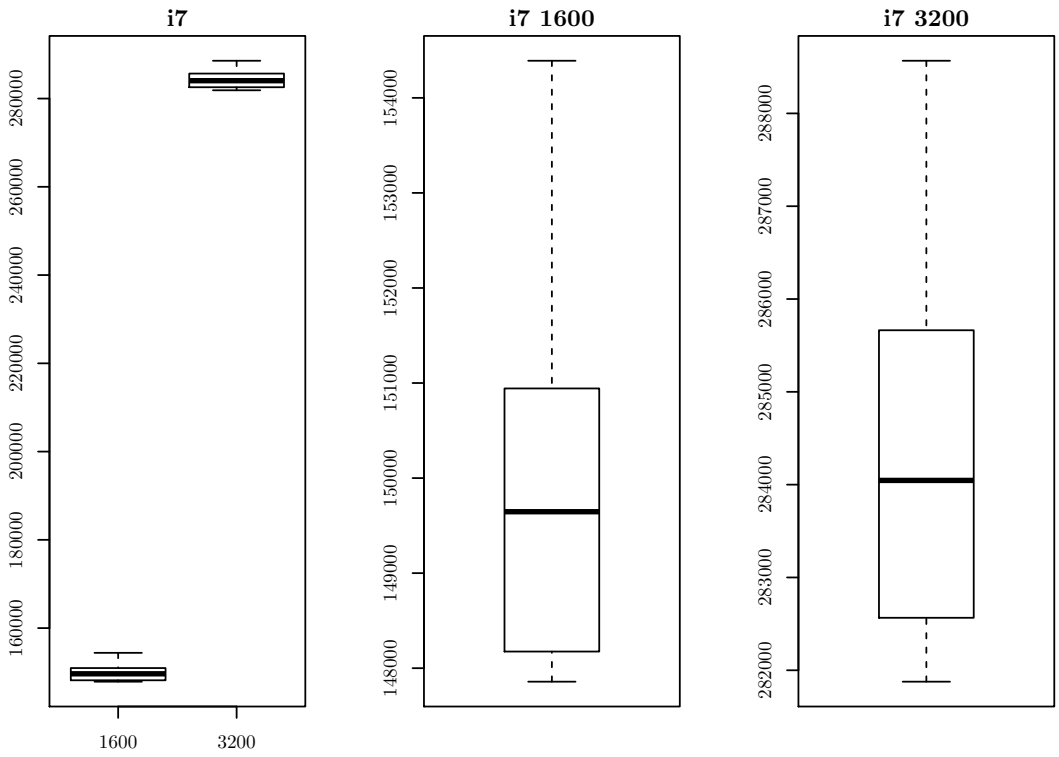


Figura 7.12: *Diagramas de caixa para os tempos de execução sequencial do AGP-I para função Rosenbrock no processador i7 após o trimming de 25%.*

Para Wilcox e Keselman (2003), a taxa de corte de 20% é o indicado na maioria dos casos. Foram testados os valores de taxa de corte de 10%, 20% e 25%. Com a taxa de corte de 25%, os pontos discrepantes não foram observados nos diagramas de caixa para os tempos de execução sequencial do AGP-I para função Rosenbrock no processador i7, como é exibido na Figura 7.12. A Tabela 7.5 apresenta um sumário dos tempos de execução sequencial do AGP-I para a função Rosenbrock após a aplicação do *trimmed mean* com taxa de corte de 25%.

Tabela 7.5: *Tempos de execução sequencial (ms) do AGP-I para a função Rosenbrock, após o trimmed mean de 25%.*

	$\overline{X}$	S	mediana	min	max	intervalo	assim.	curtose	SE
Xeon 1600	219.113,6	451,0	219.202,4	218.288	219.704	1.416,2	-0,52	-1,02	100,9
Xeon 3200	431.823,9	557,8	431.824,1	430.970	432.983	2.013,4	0,58	-0,47	124,7
i7 1600	149.801,4	1.849,8	149.646,0	147.858	154.390	6.532,4	0,79	-0,36	413,63
i7 3200	284.415,7	2.145,7	284.045,5	281.877	288.567	6.690,1	0,57	-0,98	479,8

A execução sequencial do AGP-I para a função Rosenbrock com a variação dos fatores tamanho da população e processador *multicore*, apresentou uma variabilidade dos tempos de execução no processador i7 maior do que no processador Xeon. O processador i7 possui mecanismos de otimização, como a tecnologias Intel *Turbo Boost* e o *Hyperthreading*, que podem causar a variabilidade observada. O desempenho pode ser modificado de acordo com fatores como a carga de trabalho, a temperatura do processador, o consumo de energia, dentre outros. Além da variabilidade, a distribuição dos tempos de execução no processador i7 apresentou pontos discrepantes e assimetria. Essas características dificultam a aplicação de estatísticas tradicionais e sinalizam para a necessidade de se aumentar o tamanho da amostra para se tentar modelar a variabilidade, e/ou aplicar estatísticas robustas, como o método *trimmed mean*.

Os experimentos no processador Xeon produziram tempos de execução com variabilidade menor, sem pontos discrepantes e com distribuição simétrica, facilitando a aplicação de estatísticas paramétricas tradicionais, como o teste t e a ANOVA.

7.9.2 Análise dos *Speedups*

À matriz do fatorial 2^8 foram adicionados as quatro configurações para a obtenção dos tempos sequenciais variando o tamanho da população e o tipo de processador, totalizando 260 corridas experimentais a serem testadas. As corridas referentes aos tempos sequenciais foram analisadas na seção anterior (Seção 7.9.1). O gerador de números pseudo aleatórios foi utilizado para determinar a ordem de execução das corridas e os tempos medidos foram os tempos de execução totais (*wall clock time*). O número de repetições $n = 40$ foi previamente estabelecido. O total de execuções do AGP-I foi de 10400 execuções e o tempo gasto aproximado foi de seis dias de testes.

As condições estabelecidas por Qiao et al. (2006) e Díaz-Francés e Rubio (2013) (ver Seção 6.6.2) para que a razão entre dois valores independentes e com distribuição de dados normal possa ser utilizada para estimar a razão das médias no cálculo do *speedup* foram verificadas, e as condições foram atendidas, sendo $\delta_{Y_i} < 0,23$ e $\delta_{\overline{Y_i}} < 0,04$.

A Tabela 7.6 apresenta estatísticas descritivas dos *speedups* e dos *log10(speedup)*: o tamanho n da amostra, a média, o desvio padrão S, a mediana, o valor mínimo e valor máximo, o intevalo, a assimetria e curtose da distribuição, e o erro padrão SE. A presença das estatísticas para o *log10(speedup)* será explicada mais adiante.

A Figura 7.13 apresenta, da esquerda para a direita, na parte superior, o diagrama

Tabela 7.6: *Resumo dos speedups s_p do AGP-I para a função Rosenbrock.*

	n	$\overline{Y}$	S	mediana	min	max	intervalo	assim.	curtose	SE
	256	3,21	1,07	3,18	1,35	7,69	6,34	1,14	2,74	0,07
Xeon	128	3,91	0,97	3,70	2,36	7,69	5,33	1,66	4,18	0,09
i7	128	2,51	0,61	2,53	1,35	4,30	2,95	0,45	0,76	0,05
log10	256	0,48	0,14	0,50	0,13	0,89	0,76	-0,11	0,26	0,01
Xeon	128	0,58	0,10	0,57	0,37	0,89	0,51	0,59	1,42	0,01
i7	128	0,39	0,11	0,40	0,13	0,63	0,50	-0,40	0,25	0,01

de caixas com todos os *speedups* e os *speedups* agrupados por processador. Na parte inferior estão o histograma e o gráfico de probabilidade normal dos *speedups*. A Figura 7.14 exibe esses mesmos gráficos, porém para o $log_{10}(speedup)$. A função logarítmica aplicada ao *speedup* amenizou a curtose e a assimetria da distribuição. Os diagramas de caixa mostraram menos valores discrepantes, e os pontos se aproximaram mais da reta no gráfico de probabilidade normal.

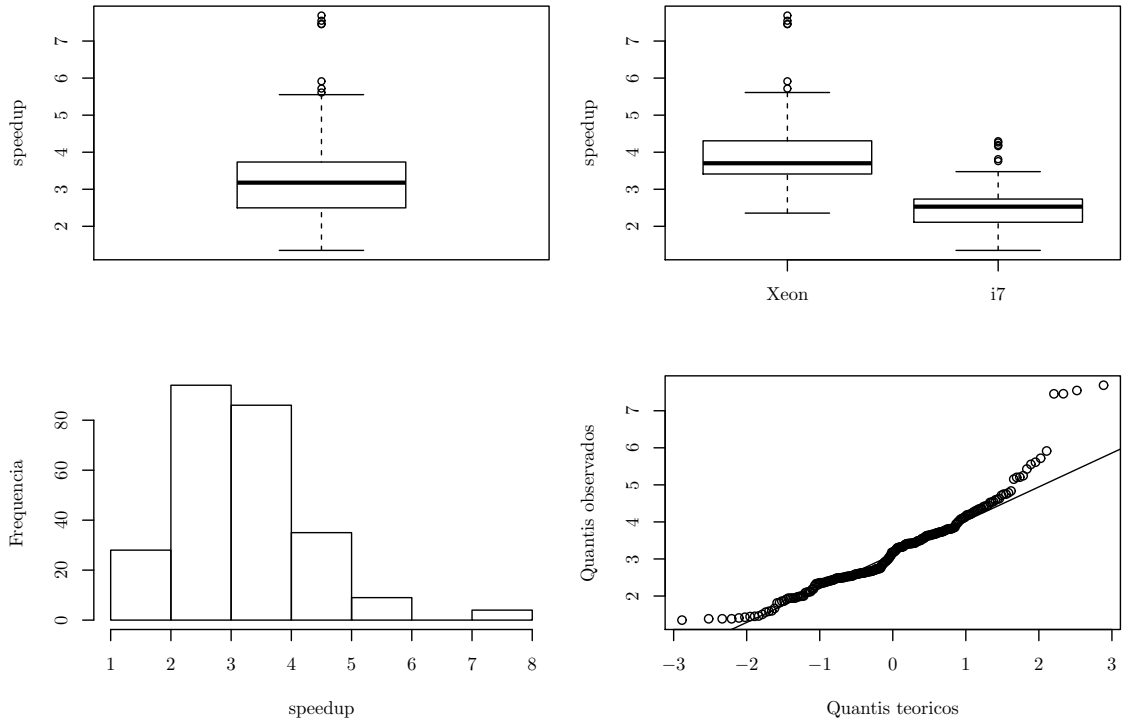


Figura 7.13: *Gráficos exploratórios dos speedups do AGP-I para função Rosenbrock.*

Os *speedups* s_p são exibidos na Figura 7.15, onde os símbolos correspondem à classificação da aceleração (ver Seção 6.5.3), e a linha tracejada separa as observações do processador Xeon e do i7. Não houve ocorrências de *speedups* sublineares e houve 14 *speedup* superlineares, todos no processador Xeon.

Os diagramas de caixa dos s_p agrupados pelos fatores estudados são mostrados na Figura 7.16. Os diagramas de caixa dos $Log_{10}(s_p)$ agrupados pelos fatores estudados

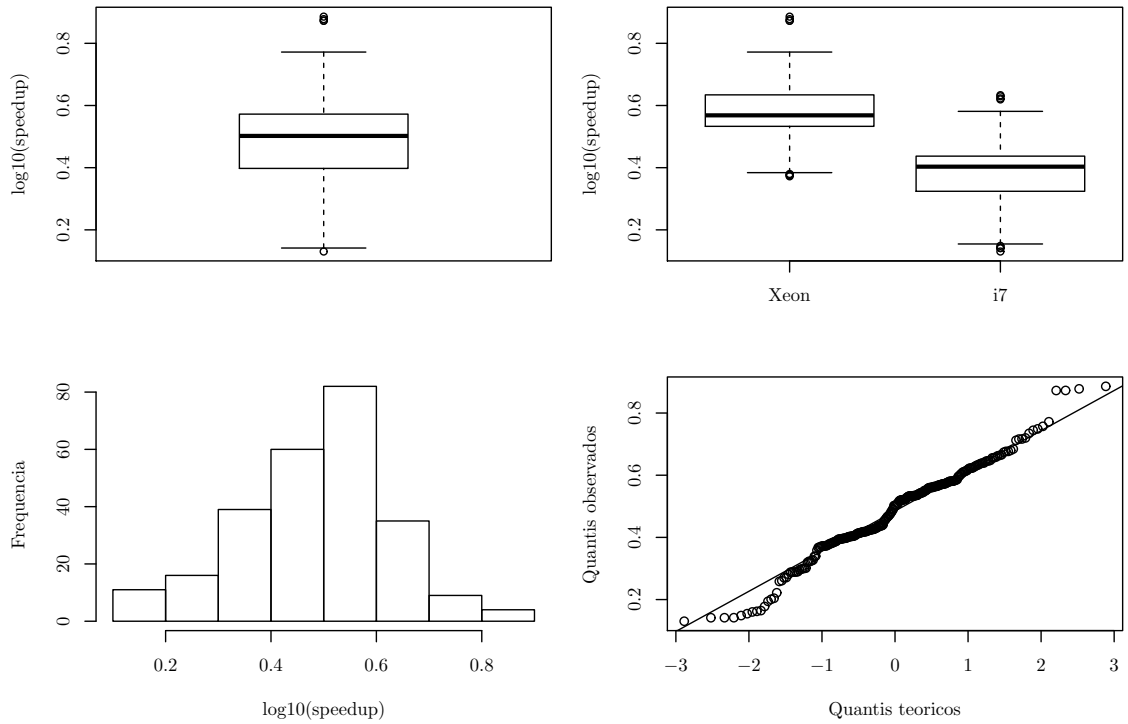


Figura 7.14: *Gráficos exploratórios dos speedups após transformação logarítmica.*

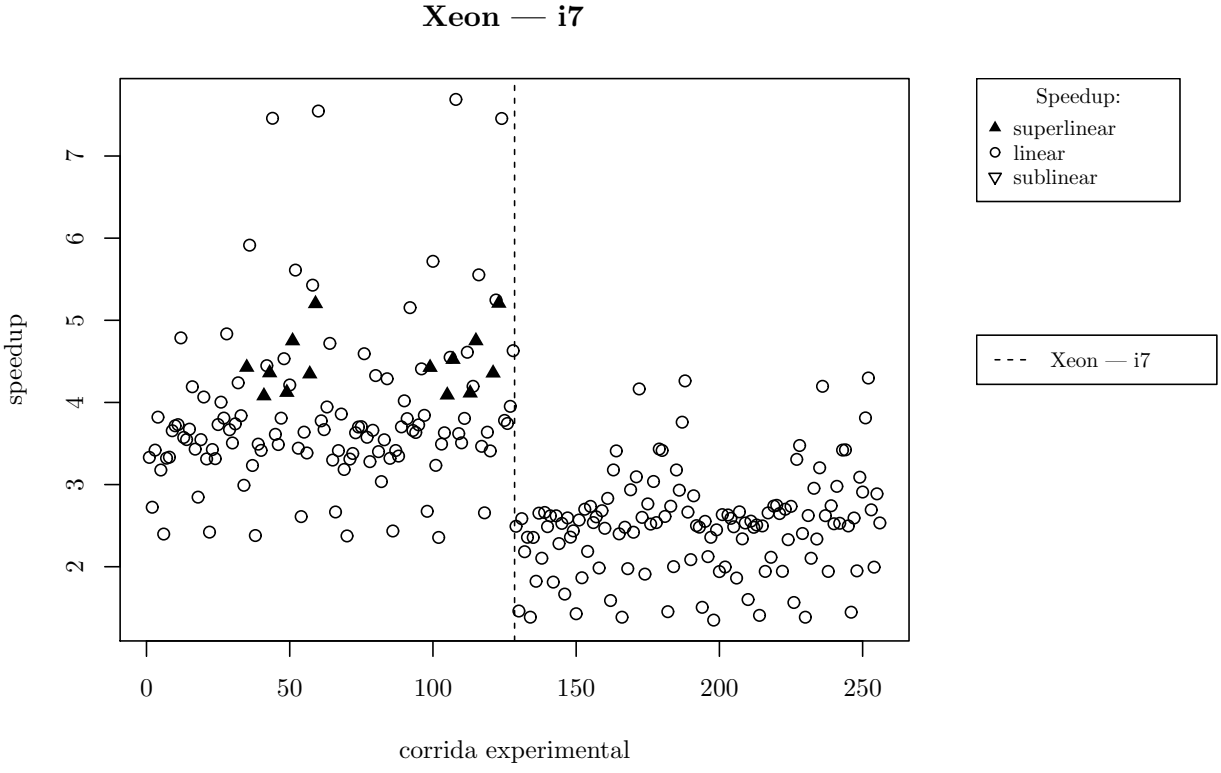


Figura 7.15: *Gráfico dos speedups s_p por corrida experimental.*

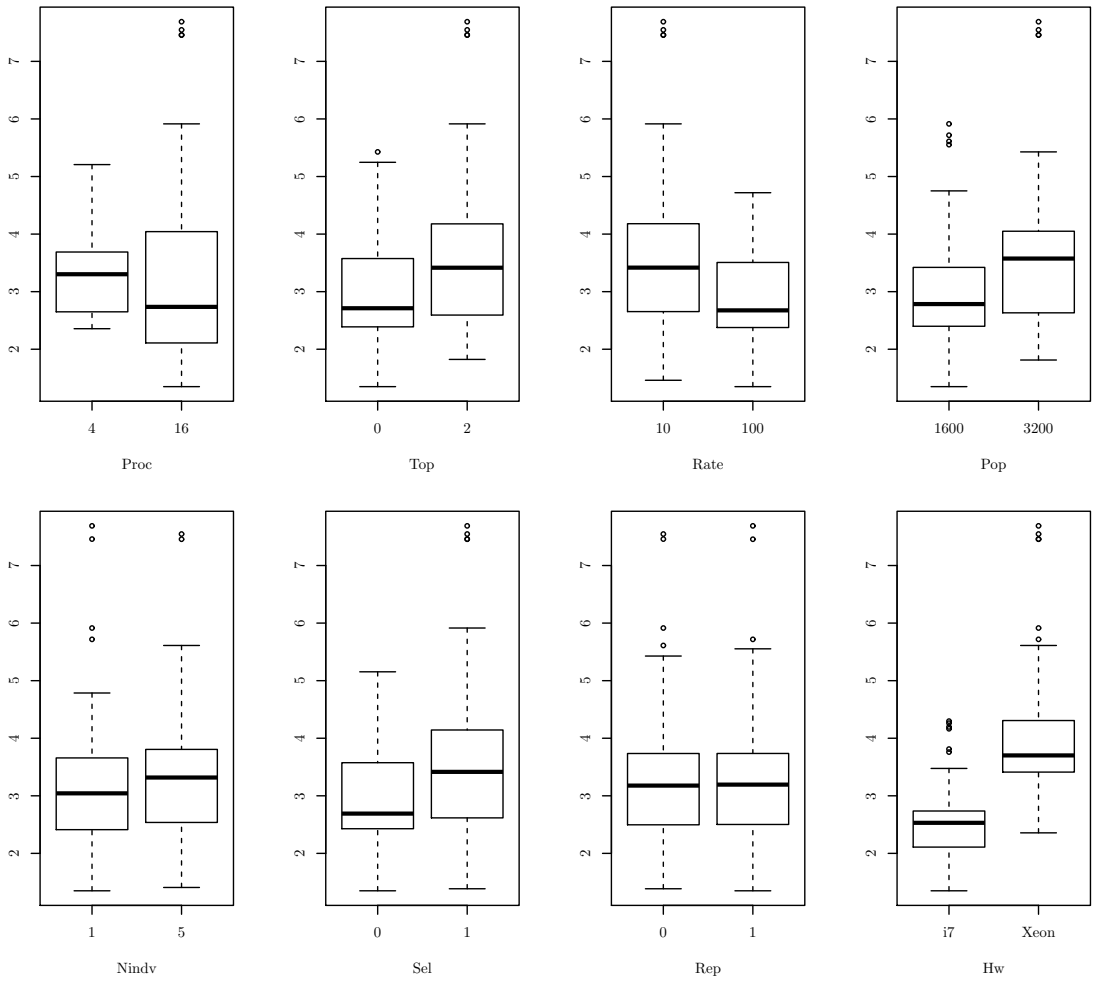


Figura 7.16: *Diagramas de caixa dos s_p agrupados pelos fatores Proc, Top, Rate, Pop, Nindv, Sel, Rep e Hw.*

são mostrados na Figura 7.17. Em ambas as figuras, os fatores Nindv e Rep quase não apresentaram distinção entre os valores de s_p para os seus níveis $-$ e $+$, enquanto que para o fator Hw foi evidente que os valores de s_p para o processador Xeon foram maiores do que para o i7.

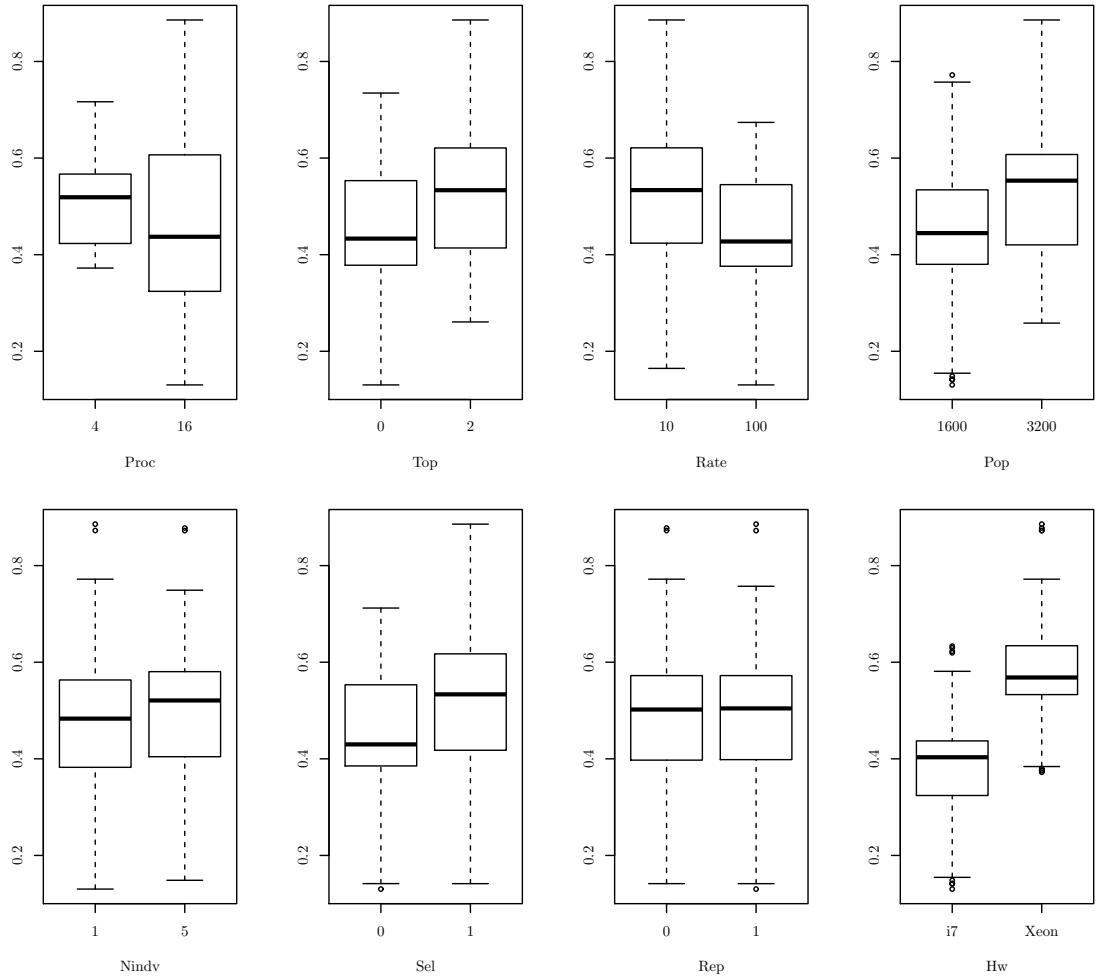


Figura 7.17: *Diagramas de caixa dos $Log_{10}(s_p)$ agrupados pelos fatores Proc, Top, Rate, Pop, Nindv, Sel, Rep e Hw.*

A análise do fatorial 2^8 sem réplicas iniciou-se com a projeção do gráfico de probabilidade normal dos efeitos estimados para a identificação dos efeitos significativos, como apresentado na Figura 7.18.

A significância estatística dos efeitos foi confirmada através do modelo com interações de ordem três. As interações de ordem maior que três são, em geral, insignificantes e podem ser usadas para estimar o erro[2]. Entretanto, a verificação dos resíduos do modelo de ordem três mostrou que a variância dos resíduos cresce à medida que os valores previstos crescem (Figura 7.19). Esse comportamento sugere a aplicação de uma trans-

[2]Princípio da esparsidade dos efeitos (Montgomery e Runger, 2009).

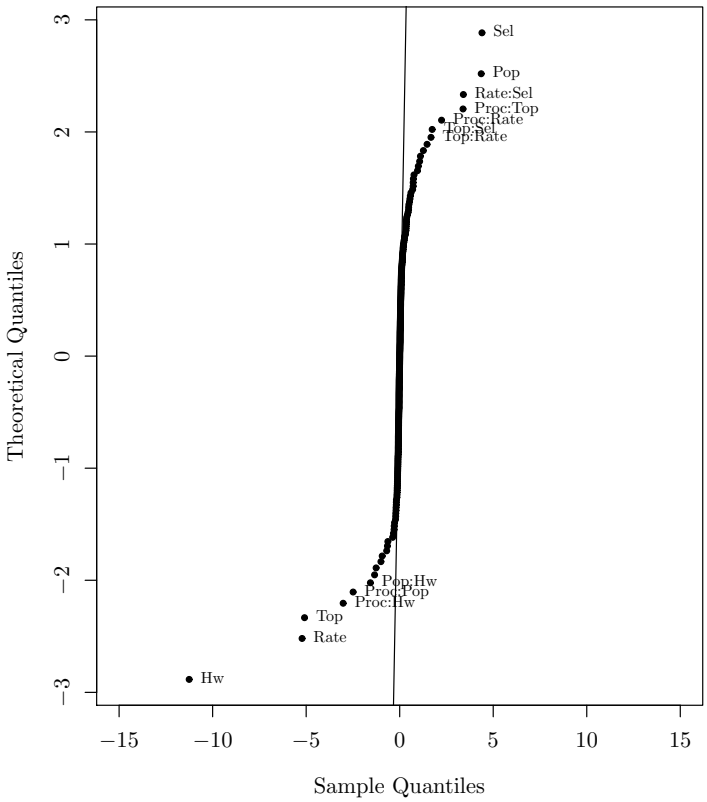


Figura 7.18: *Gráfico de probabilidade normal dos efeitos estimados do fatorial* 2^8 *dos speedups* s_p *do AGP-I para a função Rosenbrock executado nos processadores Xeon e i7.*

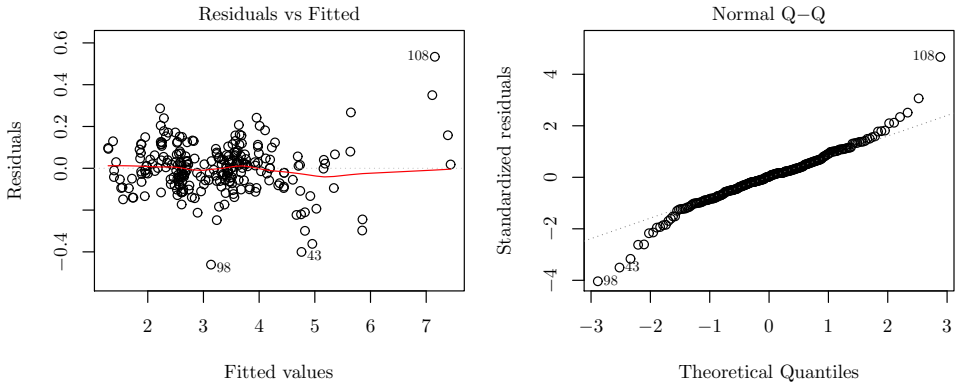


Figura 7.19: *Verificação dos resíduos: gráfico dos resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem três do fatorial* 2^8 *dos speedups do AGP-I para a função Rosenbrock.*

formação logarítmica.

A resposta dos experimentos denotada por y, foi transformada pela função logaritmo na base 10 pela seguinte equação:

$$y^* = log_{10}(y) \tag{7.3}$$

onde y^* representa a resposta transformada. As estatísticas descritivas dos $log_{10}(s_p)$ estão na Tabela 7.6.

A Figura 7.20 exibe a verificação dos resíduos do modelo de ordem três para o fatorial 2^8 após a transformação logarítmica.

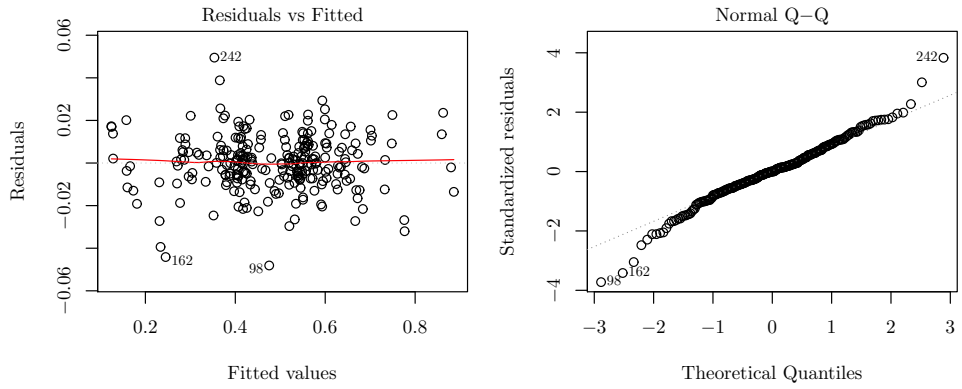


Figura 7.20: *Verificação dos resíduos do modelo após a transformação logarítmica: gráfico dos resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem três do fatorial* 2^8 *dos* $log_{10}(speedups)$ *do AGP-I para a função Rosenbrock.*

O gráfico de probabilidade normal dos efeitos estimados pelo modelo é exposto na Figura 7.21. O efeito do fator Rep permaneceu não significativo, e Rep foi removido. Assim, o fatorial 2^8 sem réplicas pode ser projetado em um fatorial 2^7 com uma réplica pelo método chamado *design projection* (Montgomery, 2009).

O passo seguinte da análise do fatorial foi executar o procedimento automático de seleção do modelo, seguindo o critério AIC (Seção 4.4.1) através da função `stepAIC` do pacote MASS do R (Venables e Ripley, 2002).

Para interpretação dos efeitos estimados pelo modelo da Tabela 7.7 é necessário reverter a transformação logarítmica aplicada, como é mostrado a seguir:

$$\widehat{y} = 10^{(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 \ldots)} = 10^{\hat{\beta}_0} \times 10^{\hat{\beta}_1 x_1} \times 10^{\hat{\beta}_2 x_2} \ldots \tag{7.4}$$

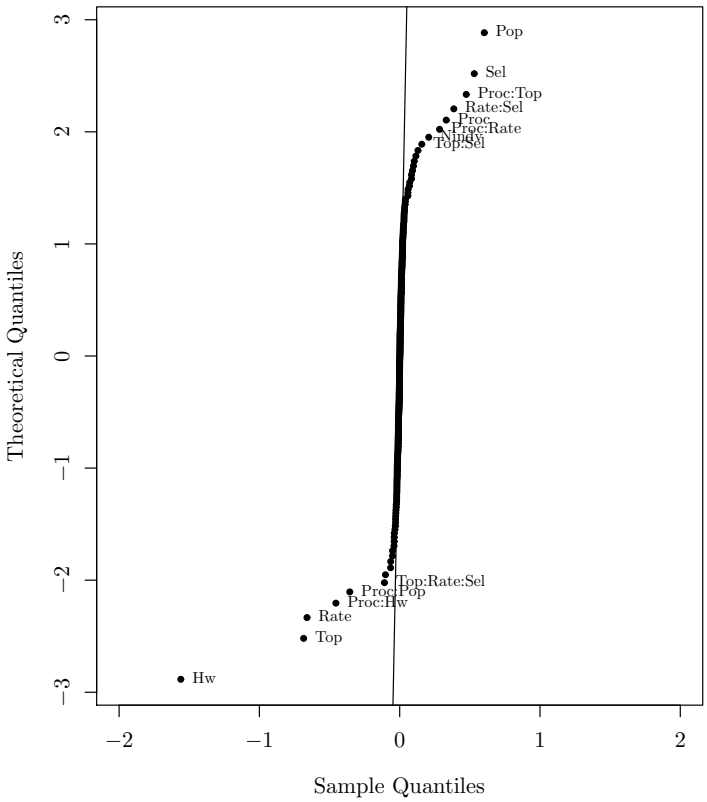


Figura 7.21: *Gráfico de probabilidade normal dos efeitos estimados do fatorial* 2^8 *dos speedups do AGP-I para a função Rosenbrock executado nos processadores Xeon e i7, após a transformação logarítmica dos speedups.*

onde $\widehat{y}$ representa a resposta estimada, $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots$ são os coeficientes estimados e $x_0, x_1, x_2, \ldots$ são as variáveis independentes codificadas.

Com a reversão da transformação logarítmica (7.4), o efeito de um fator pode ser interpretado como uma variação percentual do *speedup* médio. Por exemplo, o efeito principal do fator Hw foi de $10^{-0,195} = 0,638$. Quando o fator Hw muda do nível $-$ para o nível $+$, ou seja, de Xeon para i7, o *speedup* médio é multiplicado por $0,638$. Entretanto, no modelo (7.4) a interação de Hw com Proc foi significativa, e o efeito de Hw deve que ser interpretado em conjunto com o fator Proc, como descrito a seguir nesta seção.

Na Seção 7.4.1 foram definidos os níveis de significância adotados nos experimentos: o nível de 5% de significância estatística e significância prática para efeitos maiores que 10% do *speedup* médio. O gráfico de barras com os valores absolutos do percentual estimado de cada efeito no *speedup* médio do AGP-I para função Rosenbrock é exibido na Figura 7.22.

O modelo com os efeitos significativos (estatística e prática) é dado pela seguinte

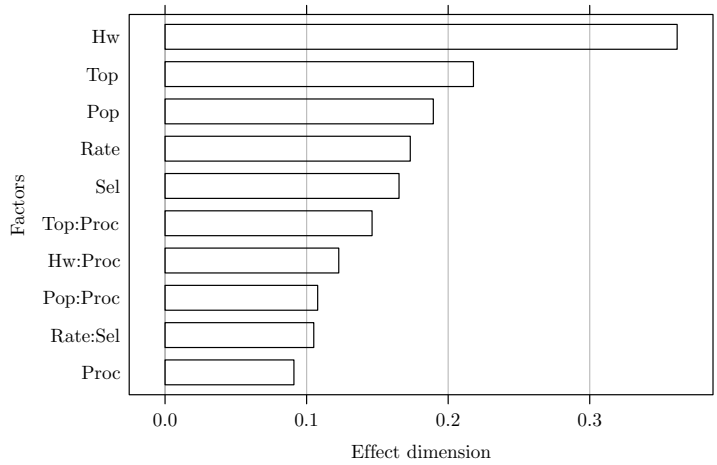


Figura 7.22: *Gráfico de barras com os efeitos estimados em valor percentual absoluto para o speedup do AGP-I para função Rosenbrock.*

equação:

$$\begin{aligned}\widehat{y^*} =& 0,484 - 0.098x_8 + 0.043x_2 + 0.038x_4 - 0.041x_3 + 0,033x_6 - 0.021x_1 + 0.030x_1x_2 \\ & - 0.028x_1x_8 + 0.022x_1x_4 - 0.024x_3x_4,\end{aligned} \tag{7.5}$$

onde as variáveis codificadas x_1, x_2, x_3, x_4, x_6 e x_8 representam os fatores Proc, Top, Rate, Pop, Sel e Hw, respectivamente. Esse modelo possui o coeficiente de determinação R^2 igual a 0,936 com 245 graus de liberdade. O mesmo modelo é apresentado em maiores detalhes na Tabela 7.7.

Tabela 7.7: *Modelo ajustado para o fatorial 2^7 do AGP-I para a função Rosenbrock, após a transformação logarítmica dos speedups.*

	Coeficiente	Erro Padrão	Valor t	Valor-p
(Interseção)	0,4837	0,0023	210,84	0,0000
Hw	-0,0975	0,0023	-42,51	0,0000
Top	0,0428	0,0023	18,66	0,0000
Pop	0,0377	0,0023	16,43	0,0000
Rate	-0,0413	0,0023	-18,01	0,0000
Sel	0,0332	0,0023	14,48	0,0000
Proc	-0,0207	0,0023	-9,04	0,0000
Top:Proc	0,0296	0,0023	12,92	0,0000
Hw:Proc	-0,0284	0,0023	-12,39	0,0000
Pop:Proc	0,0222	0,0023	9,69	0,0000
Rate:Sel	-0,0241	0,0023	-10,51	0,0000

O efeito de Proc ficou um pouco abaixo dos limite de 10%, porém Proc permaneceu no modelo (7.5) porque ele estava presente em interações significativas.

A verificação do modelo através da análise dos resíduos é apresentada na Figura 7.23. Do lado esquerdo, o gráfico dos resíduos pelos valores preditos não apresenta

estrutura ou padrão. Do lado direito, o gráfico de probabilidade normal dos resíduos exibe os pontos posicionados próximos da reta. Os testes formais confirmaram que os resíduos estavam distribuídos de forma homogênea (teste *Breusch-Pagan*, função `ncvTest()` com valor-p = 0,09) e se aproximaram da distribuição normal (teste de *Shapiro-Wilk()*, função `shapiro.test` com valor-p = 0,15). Portanto, o modelo final (7.5) foi considerado válido.

Os intervalos de confiança para os efeitos estimados para y^* e os efeitos percentuais[3] para a variável não transformada y são expostos na Tabela 7.8.

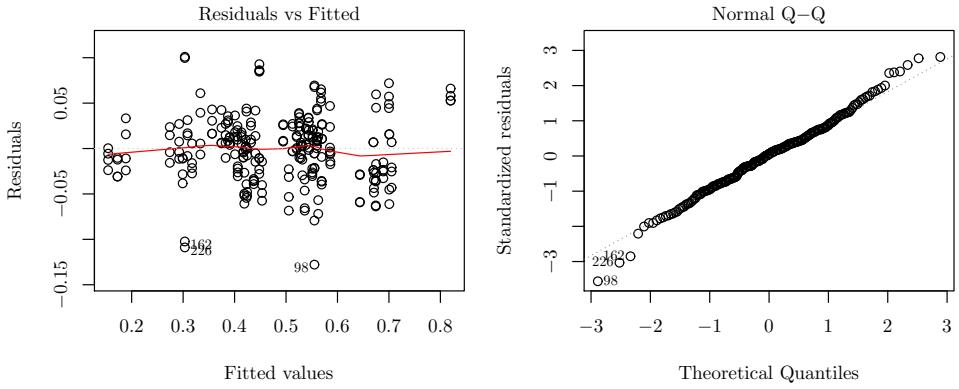


Figura 7.23: *Verificação dos resíduos do modelo final (7.5): gráfico resíduos versus preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem três do fatorial* 2^8 *dos* $log_{10}(speedups)$ *do AGP-I para a função Rosenbrock.*

Tabela 7.8: *Intervalos de confiança das estimativas de efeitos do modelo (7.5).*

Fator	Efeito	2,5 %	97,5 %	Efeito %	2,5 %	97,5 %
(Interseção)	0,48	0,48	0,49			
Hw	-0,20	-0,20	-0,19	-36%	-37%	-35%
Top	0,09	0,08	0,09	22%	20%	23%
Pop	0,08	0,07	0,08	19%	17%	20%
Rate	-0,08	-0,09	-0,07	-17%	-19%	-15%
Sel	0,07	0,06	0,08	17%	15%	20%
Proc	-0,04	-0,05	-0,03	-9%	-11%	-7%
Top:Proc	0,06	0,05	0,07	15%	12%	17%
Hw:Proc	-0,06	-0,07	-0,05	-12%	-15%	-11%
Pop:Proc	0,04	0,04	0,05	11%	10%	12%
Rate:Sel	-0,05	-0,06	-0,04	-11%	-13%	-9%

Os fatores com efeito principal significativo também participam de interações significativas e devem ser interpretados em conjunto com o fator que interage. As interações que foram significativa foram Top:Proc, Hw:Proc, Pop:Proc e Rate:Sel, e são exibidas na Figura 7.24.

[3]O efeito percentual correspondente ao $efeito$ obtido para a variável transformada é aplicado ao cálculo $10^{efeito} \times 100$.

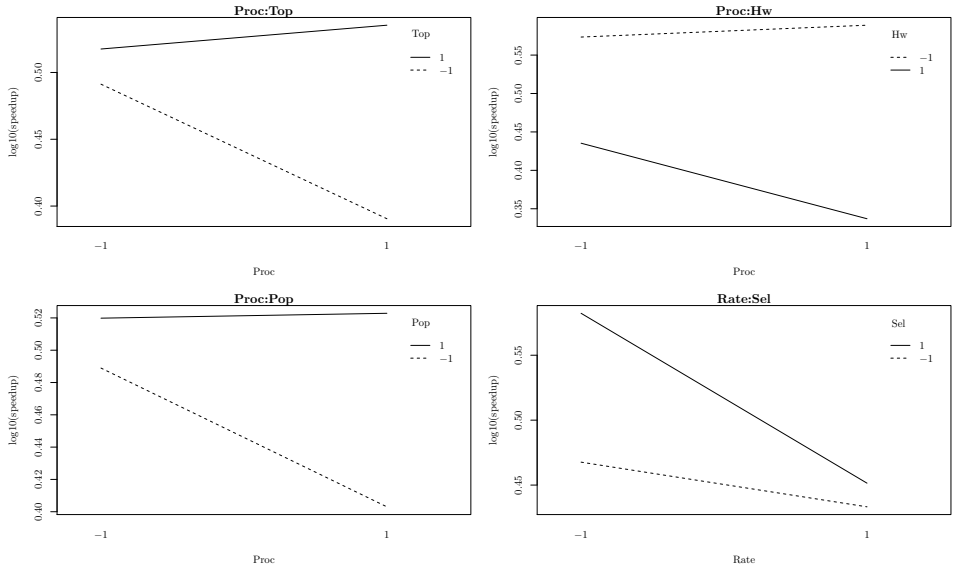


Figura 7.24: *Gráficos de interação, de cima para baixo, da esquerda para a direita: Top:Proc, Hw:Proc, Pop:Proc, Pop:Proc e Rate:Sel.*

As interações Top:Proc, Hw:Proc e Pop:Proc mostram que para se aumentar o *speedup* do AGP-I não basta aumentar o número de ilhas, pois o aumento do *speedup* também depende da quantidade de núcleos físicos disponíveis. Nos dois gráficos da parte superior e o gráfico da parte inferior, do lado esquerdo da Figura 7.24, mostram que a mudança de quatro para 16 ilhas diminuiu o $log10(s_p)$ dependendo do nível dos fatores Top, Hw e Pop.

A interação Rate:Sel revelou que mudar a estratégia de seleção de indivíduos para migrar de aleatória para a escolha dos melhores aumentou o $log10(s_p)$ quando a frequência da migração estava no nível $-$, a cada 10 gerações. Quando a frequência era de 100 gerações, a mudança da estratégia de seleção causou pouco impacto no $log10(s_p)$.

7.9.3 Análise dos *Speedups* com *Trimmed Mean*

Na seção anterior, foram verificadas as condições da aproximação da distribuição do *speedup* s_p com a distribuição normal antes da análise do planejamento fatorial 2^k. As condições descritas na Seção 6.6.2 foram satisfeitas e a análise foi realizada. Entretanto, na Seção 7.9.1 constatou-se que os tempos de execução sequencial no processador i7 continha pontos discrepantes. A média não é uma estatística robusta a pontos discrepantes. Dentre as opções para se evitar os pontos discrepantes foram exploradas a transformação logarítmica e o *trimmed mean*. A transformação não foi suficiente para melhorar a assimetria e a curtose da distribuição dos dados. Já o *trimmed mean* a 25% obteve resultados melhores.

Os *speedups* s_p foram calculados com o *trimmed mean* de 25% denotado por *trimmed* s_p, e estão exibidos na Tabela 7.9. A diferença entre as médias dos s_p e dos *trimmed* s_p não foi estatisticamente significante no teste t de *Student* com valor-p = 0,3732.

Na análise do planejamento fatorial sobre o *speedup* s_p (Seção 7.9.2), foi verificada a necessidade de se aplicar uma transformação logarítmica para estabilizar a variância dos resíduos. Da mesma forma, na análise dos *trimmed* s_p foi necessário aplicar a transformação logarítmica. A diferença entre as médias dos $log_{10}(s_p)$ e dos $log_{10}(trimmed\ s_p)$ não foi estatisticamente significante no teste t de *Student* com valor-p = 0,2652 .

O modelo obtido com os *trimmed* s_p é apresentado na Tabela 7.10, e os intervalos de confiança para os efeitos estimados estão na Tabela 7.11. O diagrama de barra com os efeitos com significância estatística e prática são mostrados na Figura 7.25.

Tabela 7.9: *Resumo dos trimmed* s_p *do AGP-I para função Rosenbrock.*

	n	média	S	mediana	min	max	intervalo	assim.	curtose	SE
	256	3,13	1,09	3,11	1,27	7,44	6,17	0,99	2,00	0,07
Xeon	128	3,89	0,94	3,68	2,30	7,44	5,14	1,54	3,83	0,08
i7	128	2,36	0,58	2,40	1,27	4,07	2,80	0,38	0,61	0,05
(log10)	256	0,47	0,15	0,49	0,11	0,87	0,77	-0,17	0,01	0,01
Xeon	128	0,58	0,10	0,57	0,36	0,87	0,51	0,48	1,34	0,01
i7	128	0,36	0,11	0,38	0,11	0,61	0,50	-0,45	0,20	0,01

Tabela 7.10: *Modelo final do trimmed* s_p *do AGP-I para função Rosenbrock.*

	Coeficiente	Erro Padrão	Valor t	Valor-p
(Interseção)	0,4692	0,0024	196,78	0,0000
Hw	-0,1099	0,0024	-46,08	0,0000
Top	0,0419	0,0024	17,56	0,0000
Pop	0,0364	0,0024	15,25	0,0000
Rate	-0,0421	0,0024	-17,66	0,0000
Sel	0,0334	0,0024	14,01	0,0000
Proc	-0,0236	0,0024	-9,88	0,0000
Top:Proc	0,0285	0,0024	11,97	0,0000
Hw:Proc	-0,0282	0,0024	-11,84	0,0000
Pop:Proc	0,0224	0,0024	9,39	0,0000
Rate:Sel	-0,0244	0,0024	-10,22	0,0000

7.9.4 Discussão

A análise do experimento teve início com análise dos tempos sequenciais. Os tempos sequenciais foram observados em quatro grupos diferentes, definidos pelo processador *multicore* e o tamanho da população: Xeon com 1600, Xeon com 3200, i7 com 1600 e i7 com 3200.

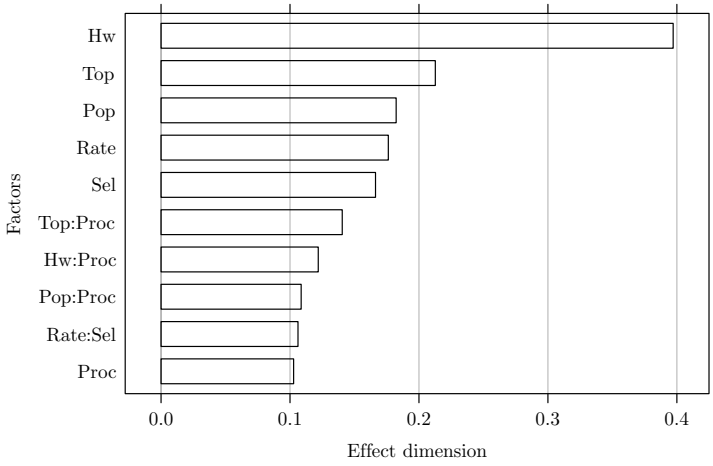


Figura 7.25: *Gráfico de barras com os efeitos estimados em valor percentual absoluto para o trimmed s_p do AGP-I para função Rosenbrock.*

Tabela 7.11: *Intervalos de confiança das estimas dos efeitos sobre o trimmed s_p.*

Fator	Efeito	2,5 %	97,5 %	Efeito %	2,5 %	97,5 %
(Interseção)	0,47	0,47	0,48			
Hw	-0,22	-0,23	-0,21	-40%	-41%	-38%
Top	0,08	0,07	0,09	21%	17%	23%
Pop	0,07	0,06	0,08	18%	15%	20%
Rate	-0,08	-0,09	-0,07	-18%	-19%	-15%
Sel	0,07	0,06	0,08	17%	15%	20%
Proc	-0,05	-0,06	-0,04	-10%	-13%	-9%
Top:Proc	0,06	0,05	0,07	14%	12%	17%
Hw:Proc	-0,06	-0,07	-0,05	-12%	-15%	-11%
Pop:Proc	0,04	0,04	0,05	11%	10%	12%
Rate:Sel	-0,05	-0,06	-0,04	-11%	-13%	-9%

As observações nos dois grupos do processador i7 apresentaram distribuição não simétrica, com cauda à direita e presença de pontos discrepantes. Ainda assim, os coeficientes de variação ficaram abaixo de 0,16, satisfazendo as condições relativas ao numerador que são necessárias para que a razão que define o *speedup* tenha uma distribuição aproximada da distribuição normal.

Foi verificado que os pontos extremos nos dois grupos do processador i7 não foram causados por uma sequência de números gerados pelo PRNG, pois o número de geração para essas observações não foi discrepante. Esses valores extremos de tempos de execução podem ter sido causados por fatores não controlados, como processos executados em segundo plano e concorrência por recursos compartilhados. Esses fatores também podem ocorrer no processador Xeon e, no entanto, os grupos de observações do Xeon não apresentaram pontos discrepantes. No i7 existem os mecanismos de otimização presentes nesse processador para maximizar o uso dos seus quatro núcleos físicos, como o mecanismo *Hyperthreading* e o *Turbo Boost*, que aumenta a frequência do *clock* dinamicamente (ver Seção 3.4). Esses fatores podem ser a razão da maior variabilidade observada nos tempos do i7.

A análise dos tempos paralelos mostrou também que a variabilidade nas observações no i7 foi maior do que no Xeon, com $\delta_{Y_i} < 0,23$ e $\delta_{\overline{Y_i}} < 0,04$. Uma vez que os tempos sequenciais e paralelos atenderam às condições para que a distribuição da razão entre as médias dos dois tempos se aproximasse da distribuição normal, os *speedups* foram calculados para cada corrida experimental. Com um speedup por corrida experimental, foi constituído um planejamento fatorial 2^8 e, em seguida, foi feita a sua análise.

A média não é uma medida robusta a pontos discrepantes. Partindo-se dessa premissa, foi realizada a análise dos *speedups* com a aplicação do método robusto do *trimmed mean*, descrito na Seção 7.9.3. As diferenças entre as médias dos s_p e dos *trimmed* s_p não foi significativa estatisticamente, e o modelo final para os *trimmed* s_p (Tabela 7.10) continha os mesmos fatores e interação com valores dos efeitos estimados muito próximos do modelo final para os s_p (Tabela 7.7).

7.10 Função Rastrigin

Nesta seção são apresentados os resultados e análises do experimento realizado para a função Rastrigin.

Com o objetivo de encontrar quais os fatores que mais afetam o *speedup* do AGP-I na resolução da função Rastrigin, foram selecionados oito fatores descritos na Tabela 7.1, sete deles relacionados aos parâmetros da migração entre ilhas e um relacionado ao processador *multicore* em que o AGP-I foi executado.

À matriz do fatorial 2^8 foram adicionadas as quatro configurações para a obtenção dos tempos sequenciais, variando o tamanho da população e o tipo de processador, totalizando 260 corridas experimentais. O PRNG foi utilizado para determinar a ordem de execução das corridas e os tempos de execução totais (*wall clock time*) foram coletados. O número de repetições $n = 40$ foi previamente estabelecido. O total de execuções do AGP-I foi de 10400 execuções e o tempo gasto aproximado foi de dois dias de testes.

7.10.1 Análise dos Tempos Sequenciais

Os tempos sequenciais foram obtidos através da execução do AGP-I com apenas uma subpopulação, e variando-se o fator Pop – tamanho da população, e o fator Hw – processador *multicore*, em dois níveis (Tabela 7.1). Os níveis da população foram de 1600 e 3200 indivíduos, e do processador *multicore* foi o Xeon ou o i7. A Tabela 7.12 apresenta um sumário dos tempos sequenciais obtidos, através do tamanho n da amostra, da média $\overline{X}$, do desvio padrão S, da mediana, do valor mínimo e valor máximo, do intervalo, da assimetria e curtose, do coeficiente de variação δ_X de X e do coeficiente de variação $\delta_{\overline{X}}$ de $\overline{X}$.

Tabela 7.12: *Tempos de execução (ms) do AGP-I sequencial para a função Rastrigin ($n = 40$).*

	$\overline{X}$	S	mediana	min	max	intervalo	assim.	curtose	δ_X	$\delta_{\overline{X}}$
Xeon 1600	4571,7	11739,0	2712,3	2621	76959	74337,60	5,86	33,15	2,57	0.41
Xeon 3200	5361,8	73,2	5365,4	5203	5507	304,13	0,05	-0,71	0,01	0,00
i7 1600	4393,2	10389,7	2529,7	1739	64275	62535,78	4,99	25,01	2,36	0,37
i7 3200	5057,8	227,7	4977,4	4825	5785	960,4	1,82	2,29	0,05	0,01

A Figura 7.26 apresenta gráficos exploratórios dos tempos sequenciais do AGP-I para a função Rastrigin. No topo, à esquerda, é exibido o histograma com uma linha

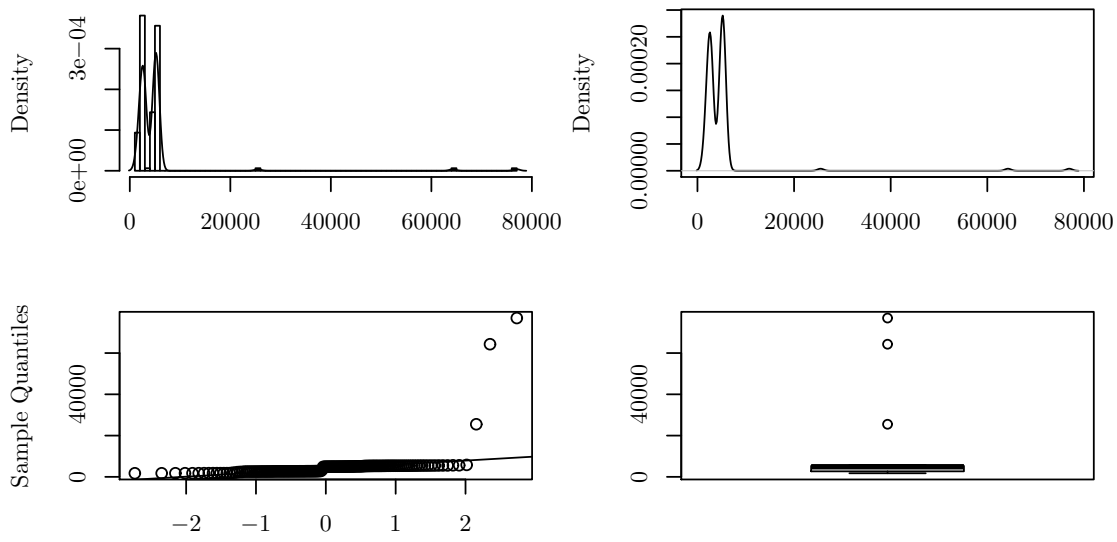


Figura 7.26: *Gráficos descritivos da distribuição dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin.*

que representa a função densidade de probabilidade estimada. No topo à direita, é apresentada a curva estimada para a função densidade. Em baixo, do lado esquerdo está o gráfico de probabilidade normal, e do lado direito está o diagrama de caixa dos tempos sequenciais. Os gráficos com histogramas e funções de densidade estimadas exibem longas caudas do lado direito, indicando a presença de valores discrepantes. O gráfico de probabilidade normal e o diagrama de caixa também exibem valores discrepantes.

Os diagramas de caixa dos tempos sequenciais são exibidos nas Figuras 7.27, 7.28 e 7.29. A Figura 7.27 exibe três gráficos com diagramas de caixa dos tempos sequenciais agrupados por: tamanho da população, processador *multicore* e ambos. As Figuras 7.28 e 7.29 apresentam os diagramas de caixa para as observações no processador Xeon e i7, respectivamente. Os diagramas de caixa exibem pontos extremos para as observações nos grupos Xeon com 1600 e i7 com 1600, e pontos discrepantes no grupo i7 com 3200 indivíduos.

A Figura 7.30 mostra os diagramas de caixas separadas por população e processador e atribui rótulos identificando os valores discrepantes. Os gráficos indicam que os valores mais extremos ocorrem para populações de 1600 indivíduos e estão associados às observações 261, 1311 e 35111. Esses pontos foram analisados e verificou-se que os números de gerações para esses casos também foram valores distantes da maioria dos outros valores, como é mostrado nos gráficos da Figura 7.31, na qual, à esquerda, o gráfico exibe os tempos sequenciais por corrida experimental e, à direita, o gráfico

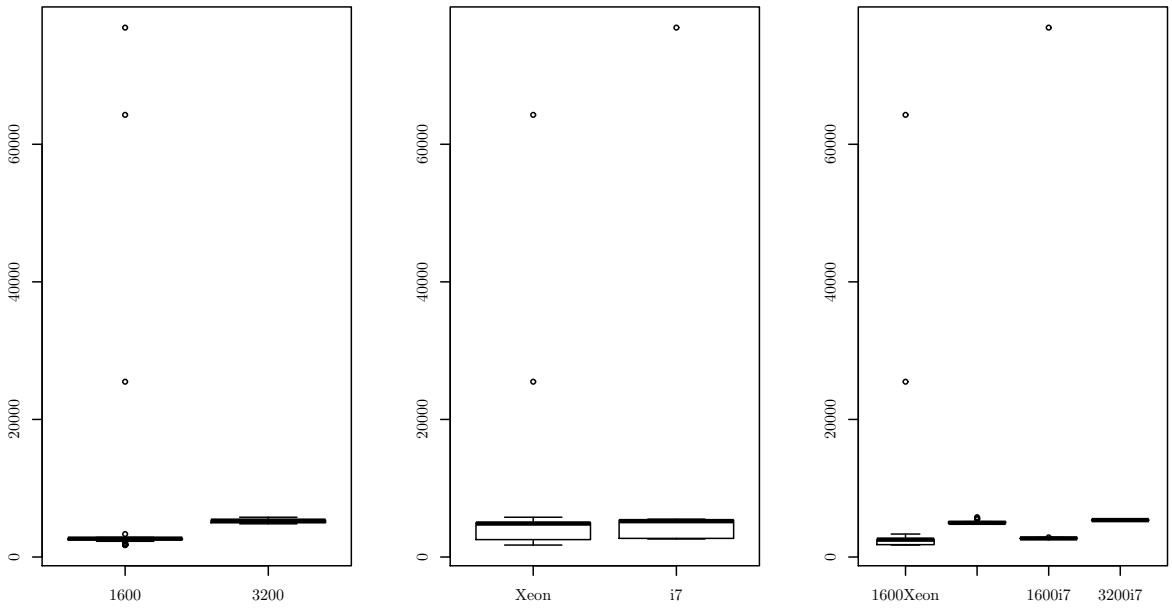


Figura 7.27: *Diagramas de caixa (boxplot) dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin agrupados por: tamanho de população, processador multicore e ambos.*

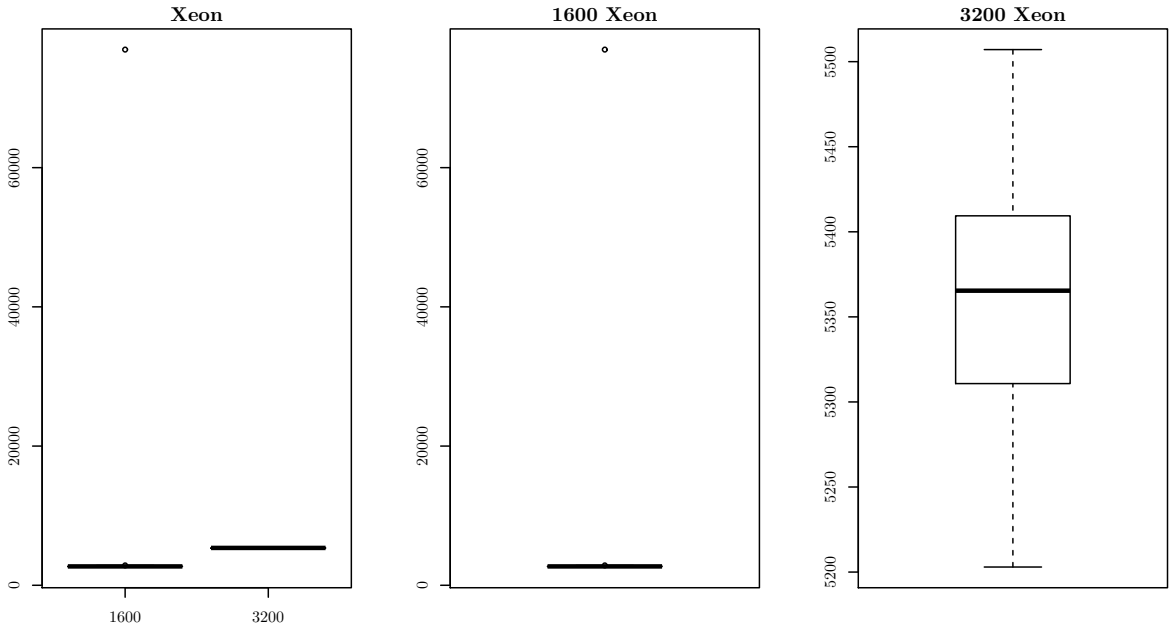


Figura 7.28: *Diagramas de caixa dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin no Xeon.*

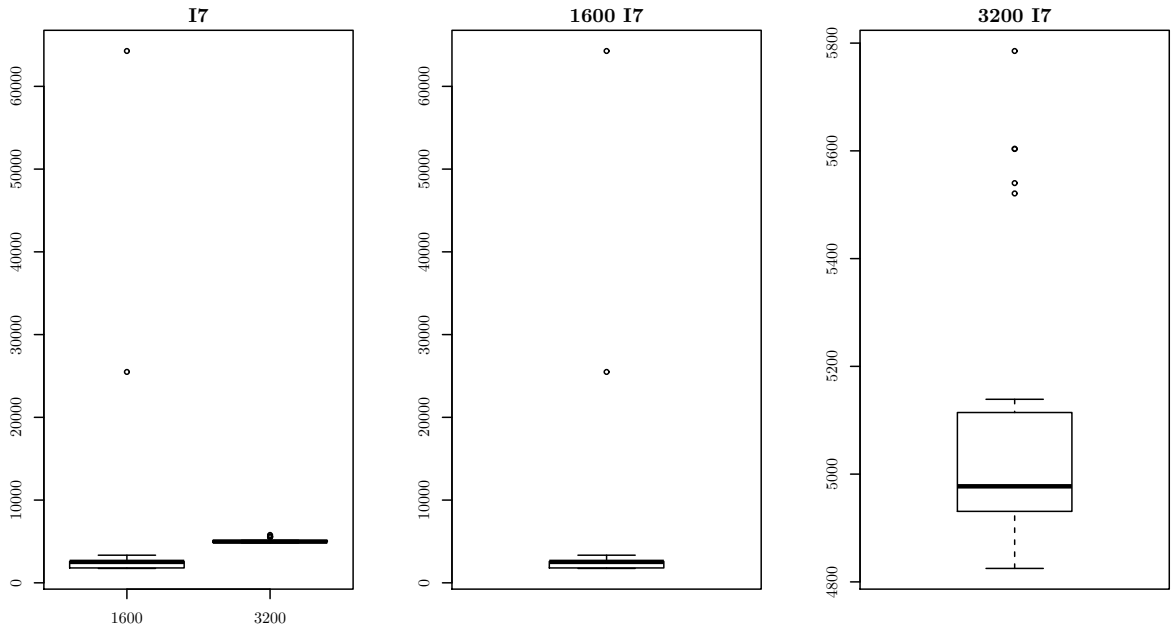


Figura 7.29: *Diagramas de caixa dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin no i7.*

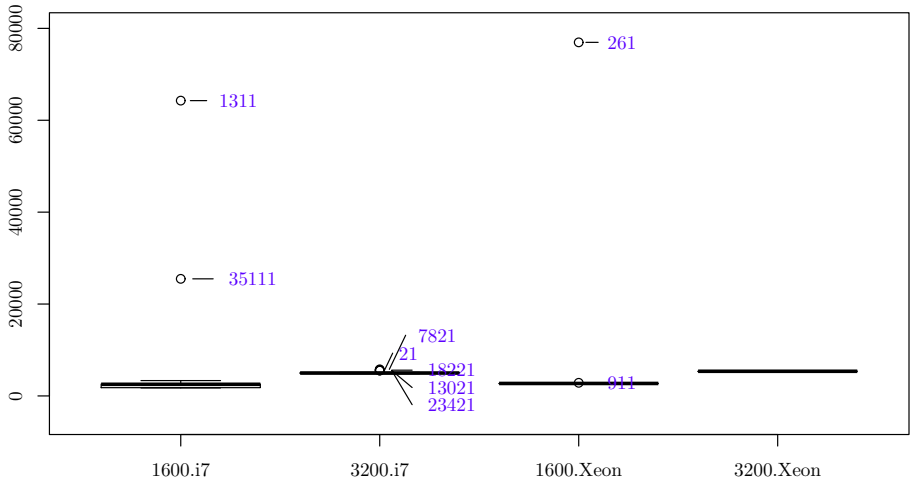


Figura 7.30: *Diagramas de caixa dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin com tempos de execução agrupados por tamanho de população e processador multicore. Os rótulos numéricos identificam as observações que geraram os pontos discrepantes mostrados nos diagramas.*

exibe a quantidade de gerações executadas por corrida experimental. As observações 261, 1311 e 35111 obtiveram valores discrepantes nas medidas de tempo de execução e de quantidade de gerações. Para essas observações o PRNG produziu uma sequência que fez com que AGP-I demorasse mais para conseguir chegar ao mínimo da função e, portanto, esses três pontos discrepantes não foram fruto de medidas espúrias ou erros de medição.

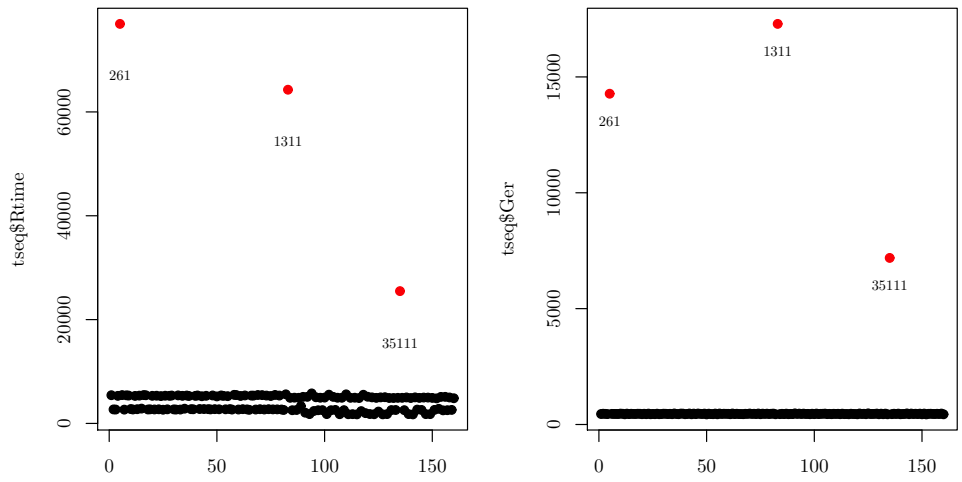


Figura 7.31: *Gráfico dos tempos sequenciais (ms) por corrida experimental e gráfico da quantidade de gerações por corrida experimental do AGP-I sequencial para a função Rastrigin. Os rótulos numéricos identificam as observações que geraram os pontos discrepantes mostrados nos gráficos.*

Pontos discrepantes podem mascarar a presença de outros discrepantes. Por exemplo, se uma discrepância é grande e outra é moderada, o desvio padrão será muito afetado pelo discrepante grande, fazendo com que o discrepante moderado pareça relativamente normal. Assim que o discrepante grande é removido, o desvio padrão diminui, e o discrepante moderado passa a parecer não usual. Ao remover os três pontos discrepantes com rótulos 261, 1311 e 35111, outros discrepantes tornaram-se mais evidentes, como é exibido na Figura 7.32.

A separação manual dos pontos discrepantes, sua identificação, verificação e remoção, para um número reduzido de pontos é fácil. Porém, quando o número de variáveis, níveis e a quantidade de réplicas é grande, o volume de dados aumenta e inviabiliza a separação manual, demandando a aplicação de um método automático.

A transformação dos dados é um método automático que pode amenizar os pontos discrepantes. A transformação log_{10} foi aplicada aos tempos sequenciais. Os diagramas de caixa dos tempos transformados é apresentada na Figura 7.33. A transformação dos

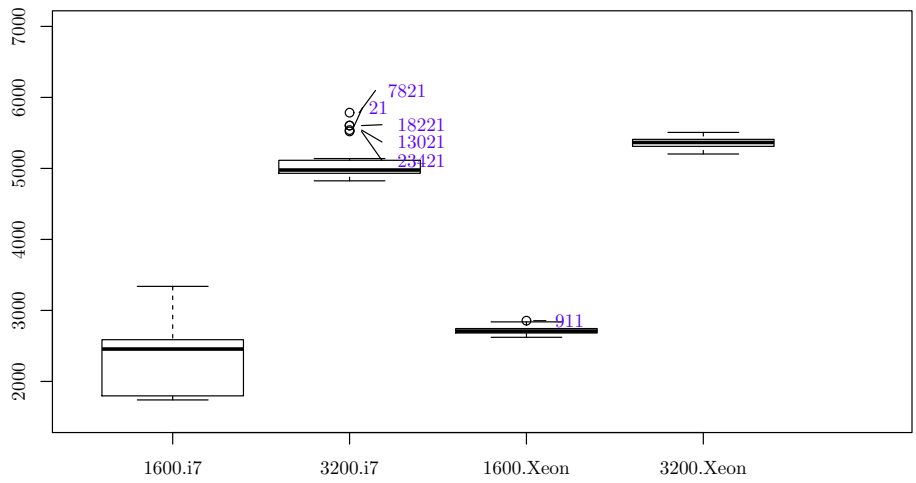


Figura 7.32: *Diagramas de caixa dos tempos da execução sequencial (ms) do AGP-I após remover os pontos discrepantes* 261, 1311 *e* 35111, *outros pontos aparecem como discrepantes. Os rótulos numéricos identificam as observações que geraram os pontos discrepantes mostrados nos diagramas.*

dados não foi suficiente para eliminar os pontos discrepantes.

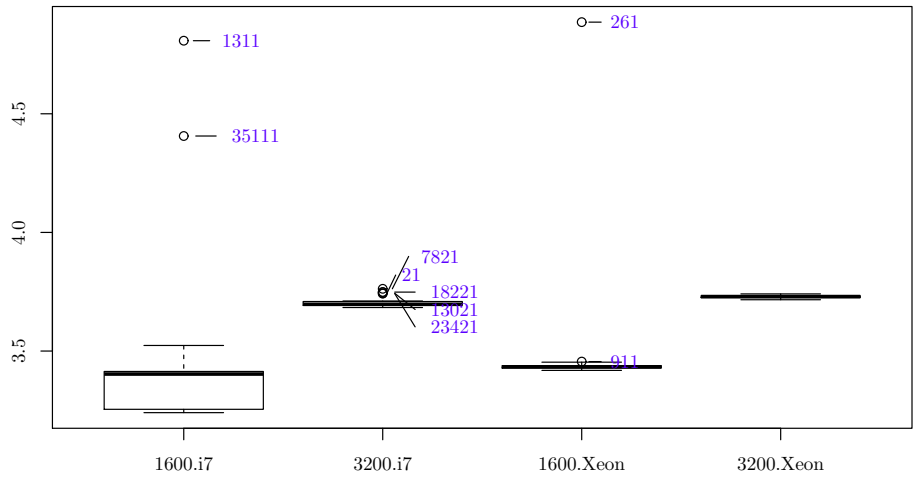


Figura 7.33: *Diagramas de caixa dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin após transformação por* log_{10}, *e com tempos de execução agrupados por: tamanho de população e processador multicore. Os rótulos numéricos identificam as observações que geraram os pontos discrepantes mostrados nos diagramas.*

Métodos robustos proveem um modo automático de detectar, minimizar (ou remover) e identificar pontos discrepantes. O *trimmed mean* é um método robusto (Wilcox e Keselman, 2003). Inicialmente foi testado o *trimmed mean* com taxa de corte de 10%, para o qual os coeficientes de variação δ_X e $\delta_{\overline{X}}$ ficaram abaixo de 0,2 (Tabela 7.13), e não foram observados pontos discrepantes extremos nos diagramas de caixa dos tempos de execução (Figura 7.34). A condição $\delta_X < 1$ necessária para a aproxima-

ção da distribuição dos *speedups* da distribuição normal, segundo Díaz-Francés e Rubio (2013), foi satisfeita e outros valores de taxa de corte não foram testados.

Na Figura 7.34, o diagrama de caixa para o i7 com 1600 indivíduos apresenta uma variabilidade maior, e o diagrama de caixa para o i7 com 3200 indivíduos exibe um ponto discrepante.

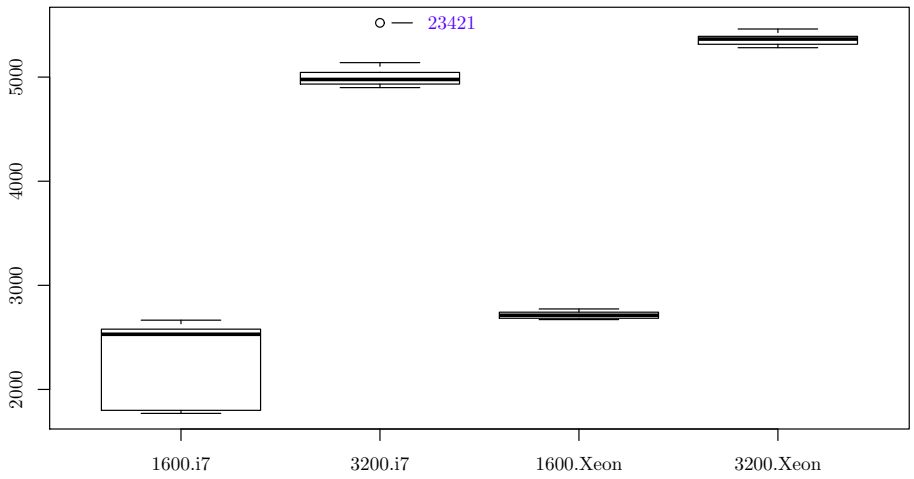


Figura 7.34: *Diagramas de caixa dos tempos de execução sequencial (ms) do AGP-I para a função Rastrigin com tempos de execução após aplicação do trimming de 10% agrupados por: tamanho de população e processador multicore. Os rótulos numéricos identificam as observações que geraram os pontos discrepantes mostrados nos diagramas.*

A Tabela 7.13 apresenta as estatísticas descritivas dos tempos sequenciais após o *trimmed mean* de 10% dos valores em cada extremidade de cada um dos quatro grupos de dados: Xeon com 1600, Xeon com 3200, i7 com 1600 e i7 com 3200. Os valores das estatísticas relacionadas a variabilidade dos dados, como os coeficientes de variação δ_X e $\delta_{\overline{X}}$, erro padrão, assimetria e curtose diminuiriam.

Tabela 7.13: *Tempos de execução (ms) do AGP-I sequencial para a função Rastrigin após trimmed mean de 10% ($n = 32$).*

	$\overline{X}$	S	mediana	min	max	intervalo	assim.	curtose	δ_X	$\delta_{\overline{X}}$
Xeon 1600	2714,3	30,6	2712,3	2671	2773	102,2	0,21	-1,24	0,01	0,00
Xeon 3200	5362,0	52,0	5365,4	5282	5462	179,7	0,31	-0,95	0,01	0,00
i7 1600	2275,2	361,1	2529,7	1770	2665	895,3	-0,45	-1,70	0,16	0,03
i7 3200	5010,0	119,0	4977,4	4899	5521	621,9	2,48	7,91	0,02	0,00

A execução sequencial do AGP-I para a função Rastrigin, com a variação dos fatores tamanho da população e processador *multicore*, comprovou o efeito significativo das mudanças de níveis desses dois fatores.

Apesar dos três pontos extremos terem sido fruto das sequências de números gerados pelo PRNG, nem todos os pontos discrepantes observados podem ser justificados dessa forma. Os tempos sumarizados na Tabela 7.13 mostram que para o processador i7 e Pop = 1600 o intervalo entre o tempo máximo e o tempo mínimo, 1,7 a 2,7 segundos, respectivamente, é de aproximadamente um segundo. A variabilidade causada pelos processos executados em segundo plano, pela concorrência por recursos compartilhados, pelos mecanismos de otimização do processador, dentre outros, são proporcionalmente maiores para os tempos de execução pequenos, como os do AGP-I para a função Rastrigin. Os tempos de execução pequenos são mais mais sensíveis à influência dos fatores não controlados do ambiente de execução.

7.10.2 Análise dos *Speedups*

As condições estabelecidas por Qiao et al. (2006) e Díaz-Francés e Rubio (2013) (ver Seção 6.6.2) para que a razão das médias $\overline{X}/\overline{Y}$ que determina o *speedup* tenha uma distribuição aproximada com a distribuição normal foram verificadas. A condição $\delta_X < 1$, estabelecida por Díaz-Francés e Rubio (2013), não foi satisfeita pelos tempos de execução no Xeon com 1600 e nem no i7 com 1600. A condição $\delta_{\overline{Y}} < 0,2$, de Qiao et al. (2006), também não foi satisfeita, e, portanto, não foi possível garantir que a distribuição dos *speedups* seja próxima da distribuição normal.

Uma solução automática para diminuir o coeficiente de varição é a aplicação de um método robusto, como o *trimmed mean*. O método *trimmed mean* foi aplicado variando-se a taxa de corte de 10% até 35%. Os resultados tiveram coeficientes de variação menores, mas ainda assim não foi possível atender às condições estabelecidas por Qiao et al. (2006) e Díaz-Francés e Rubio (2013).

De acordo com Qiao et al. (2006), um estimador para o tamanho da amostra necessário para obter $\delta_{\overline{Y}} < 0,2$ é dado pela Equação (6.11). O tamanho da amostra estimado para o processador Xeon foi $n_s > 624$ e para o i7, $n_s > 290$. O número de réplicas foi aumentado e a análise dos resultados está descrita na próxima seção.

7.10.3 Análise dos *Speedups* com Aumento do Tamanho da Amostra

Para manter os dados balanceados com a mesma quantidade de observações em cada grupo, o fatorial completo com 1001 réplicas foi executado no processador Xeon. O processador i7 não estava disponível para testes dessa dimensão. O tempo gasto nos

testes no Xeon foi de aproximadamente 19 dias.

A análise dos fatorial completo com 1001 réplicas foi iniciada com a análise dos tempos de execução sequencial, para os quais foram estimadas as estatísticas descritivas apresentadas na Tabela 7.14: média $\overline{X}$, desvio padrão S, mediana, mínimo, máximo, intervalo, assimetria, curtose, erro padrão SE, coeficiente de variação δ_X de X e o coeficiente de variação $\delta_{\overline{X}}$ de $\overline{X}$. Os tempos sequenciais para população com 1600 indivíduos apresentou pontos discrepantes, como mostra a Figura 7.35.

Tabela 7.14: *Estatísticas descritivas dos tempos de execução sequencial (ms) do AGP-I para função Rastrigin no Xeon com* $n = 1001$.

	$\overline{X}$	S	mediana	min	max	intervalo	assim.	curtose	SE	δ_X	$\delta_{\overline{X}}$
1600	67670.0	33892.0	2698,7	2544	551307	548763,4	10,89	132,82	1071,2	5,01	0,16
3200	5358,4	96,4	5351,1	5111,7	6516,3	1404,6	5,00	54,27	3,1	0,02	0,00

O aumento do tamanho da amostra para populações com 1600 indivíduos diminuiu o $\delta_{\overline{X}_{1600}}$, mas o $\delta_{X_{1600}}$ permaneceu bem acima de um, $\delta_{X_{1600}} \leq 5,01$. A presença dos pontos discrepantes afetou a média $\overline{X}$, que ficou muito acima da mediana.

Os tempos de execução paralela apresentaram $\delta_{Y_i} > 1$, sendo o maior valor $\delta_{Y_i} = 5,09$. Foram identificados pontos discrepantes nos tempos de execução associados às corridas experimentais que obtiveram valor de $\delta_Y > 1$. O método do *trimmed mean* foi aplicado para amenizar os pontos discrepantes.

7.10.4 Análise dos *Speedups* com *Trimmed Mean*

A aplicação do *trimmed mean* de 10% eliminou os pontos discrepantes das execuções sequenciais do AGP-I. Os diagramas de caixa dos tempos de execução sequencial mostrados na Figura 7.36 não apresentam pontos discrepantes. A Tabela 7.15 exibe as estatísticas descritivas dos tempos sequenciais do AGP-I após o *trimmed mean* de 10%: média $\overline{X}$, desvio padrão S, mediana, mínimo, máximo, intervalo, assimetria, curtose, erro padrão SE, coeficiente de variação δ_X de X e o coeficiente de variação $\delta_{\overline{X}}$ de $\overline{X}$.

Tabela 7.15: *Estatísticas descritivas dos tempos de execução sequencial (ms) do AGP-I para função Rastrigin no Xeon após o trimmed mean de 10% e* $n = 801$.

	$\overline{X}$	S	mediana	min	max	intervalo	assim.	curtose	SE	δ_X	$\delta_{\overline{X}}$
1600	2700,83	28,99	2698,74	2647,64	2765,55	117,91	0,15	-0,84	1,02	0,01	0,00
3200	5352,16	44,50	5351,13	5269,60	5453,59	183,99	0,14	-0,88	1,57	0,01	0,00

Os tempos paralelos do AGP-I após o *trimmed mean* de 10% apresentaram cinco casos em que $\delta_{Y_i} > 1$ e nenhum com $\delta_{\overline{Y_i}} > 0,2$, sendo $\delta_{Y_i} < 1,53$ e $\delta_{\overline{Y_i}} < 0,05$. De acordo com Qiao et al. (2006), se $\delta_{\overline{Y}} < 0,2$, então a distribuição da razão $\overline{X}/\overline{Y}$ se

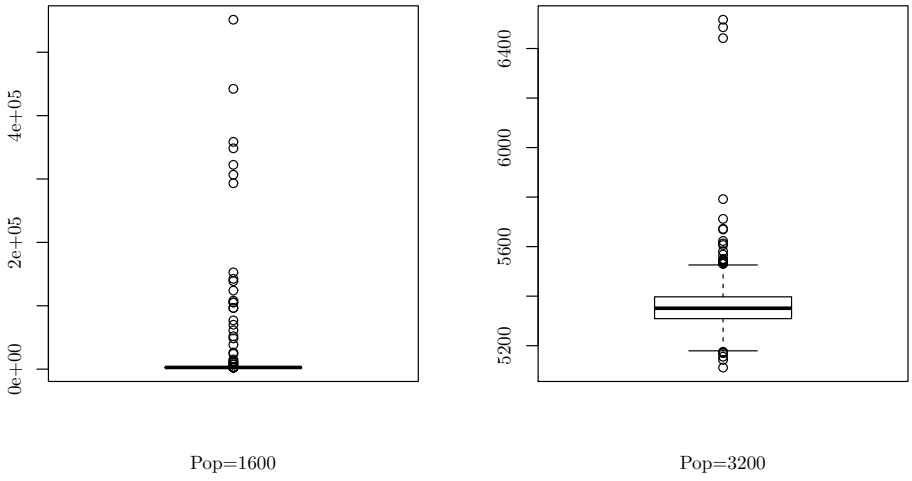


Figura 7.35: *Diagrama de caixa dos tempos de execução (ms) sequencial do AGP-I para função Rastrigin com 1000 réplicas.*

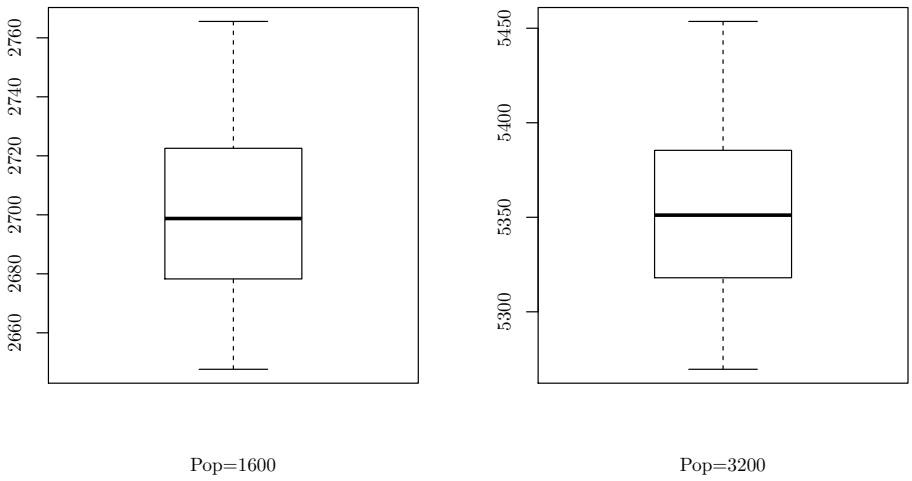


Figura 7.36: *Diagrama de caixa dos tempos de execução sequencial (ms) do AGP-I para função Rastrigin após o trimmed mean de 10%.*

aproxima da distribuição normal. Os *speedups* foram então calculados como uma razão ponderada (6.10). A Figura 7.37 apresenta os gráficos exploratórios dos *trimmed* s_p.

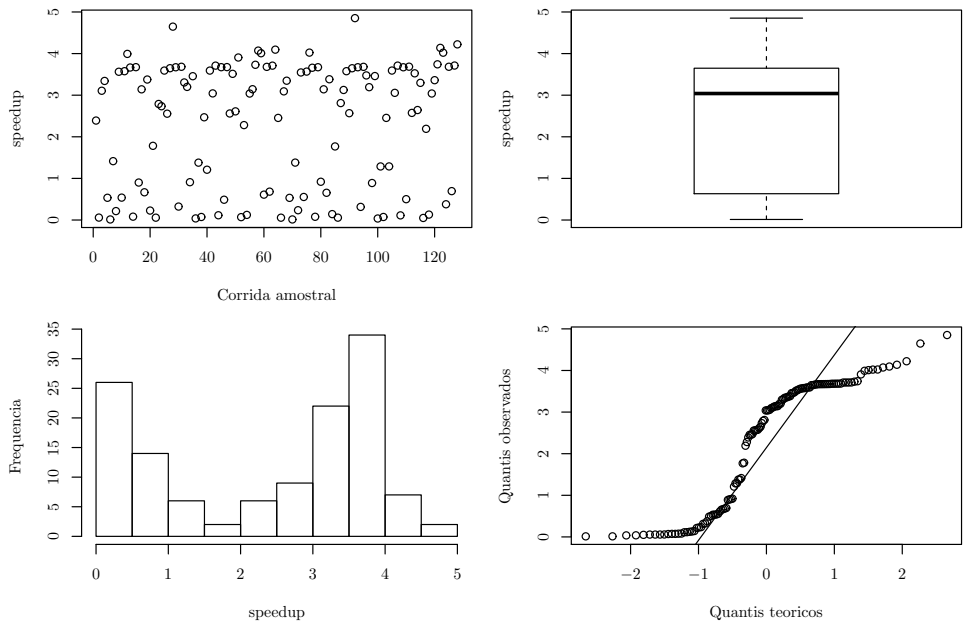


Figura 7.37: *Gráficos exploratórios dos trimmed s_p do AGP-I para função Rastrigin, no Xeon com $n = 801$.*

Os *trimmed* s_p são exibidos na Figura 7.38, onde os símbolos correspondem à classificação do *speedup*: sublinear, linear e superlinear (ver Seção 6.5.3).

A Tabela 7.16 apresenta as estatísticas descritivas para os *speedups* depois do *trimmed mean* de 10%, denotados por *trimmed* s_p.

Tabela 7.16: *Speedups do AGP-I para Rastrigin depois do trimmed mean de 10% (trimmed s_p).*

n	média	S	mediana	min	max	intervalo	assim.	curtose	SE
128	2,32	1,50	3,04	0,01	4,85	4,84	-0,40	-1,47	0,13

Os diagramas de caixa dos *trimmed* s_p agrupados pelos fatores estudados são mostrados na Figura 7.39. Os fatores Proc e Pop apresentaram maior distinção entre os valores dos *speedups* para os níveis $-$ e $+$, e o fator Sel quase não mostrou distinção.

A análise do planejamento fatorial 2^7 sem réplicas iniciou-se com a identificação dos efeitos significativos, através da projeção do gráfico de probabilidade normal dos efeitos estimados (Figura 7.40). A significância estatística dos efeitos foi confirmada através do modelo com interações de até ordem três. As interações de ordem maior que três são, em geral, insignificantes e podem ser usadas para estimar o erro[4].

[4]Princípio da esparsidade dos efeitos (Montgomery e Runger, 2009).

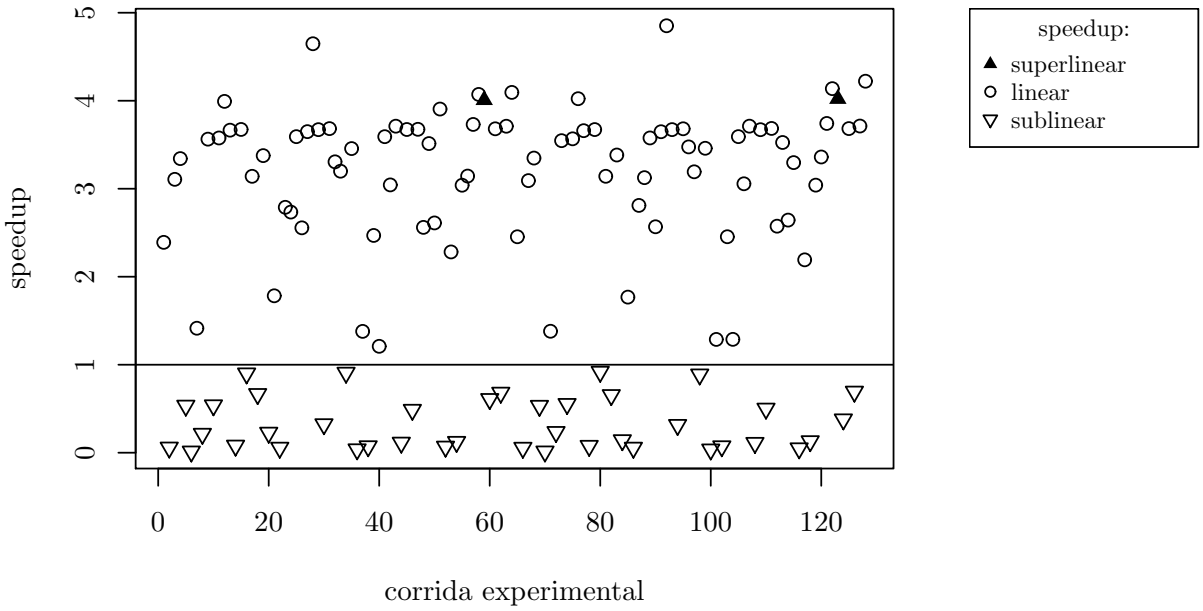


Figura 7.38: *Gráfico dos trimmed s_p por corrida experimental.*

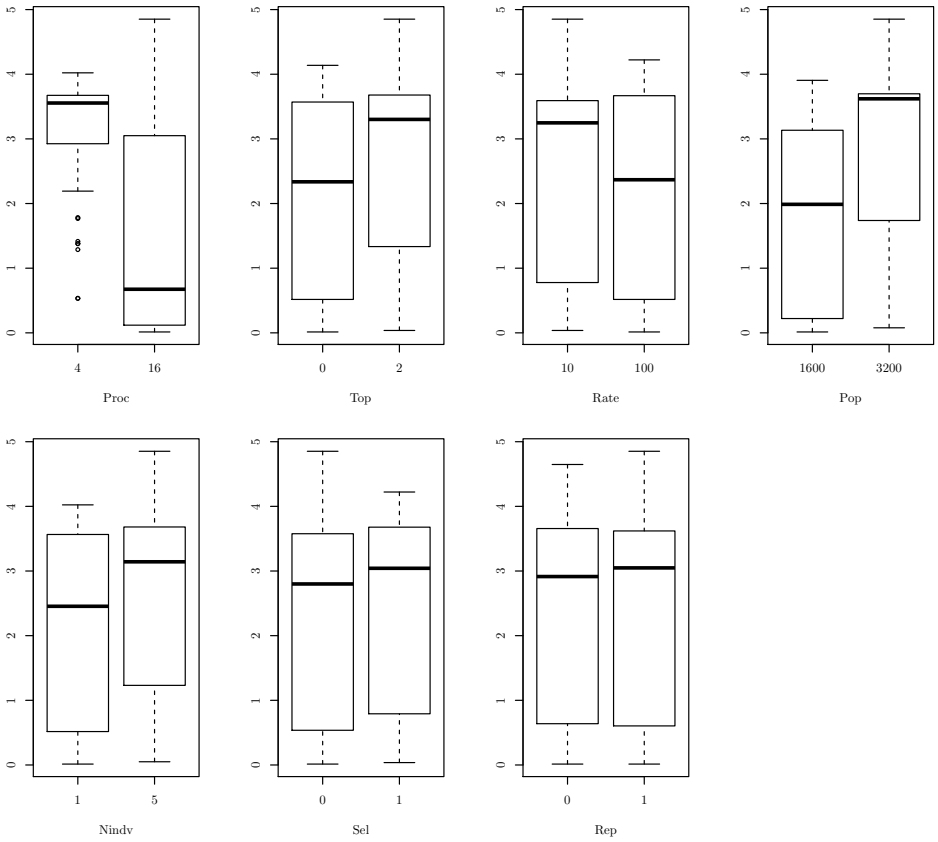


Figura 7.39: *Diagramas de caixa dos trimmed s_p agrupados pelos fatores Proc, Top, Rate, Pop, Nindv, Sel e Rep.*

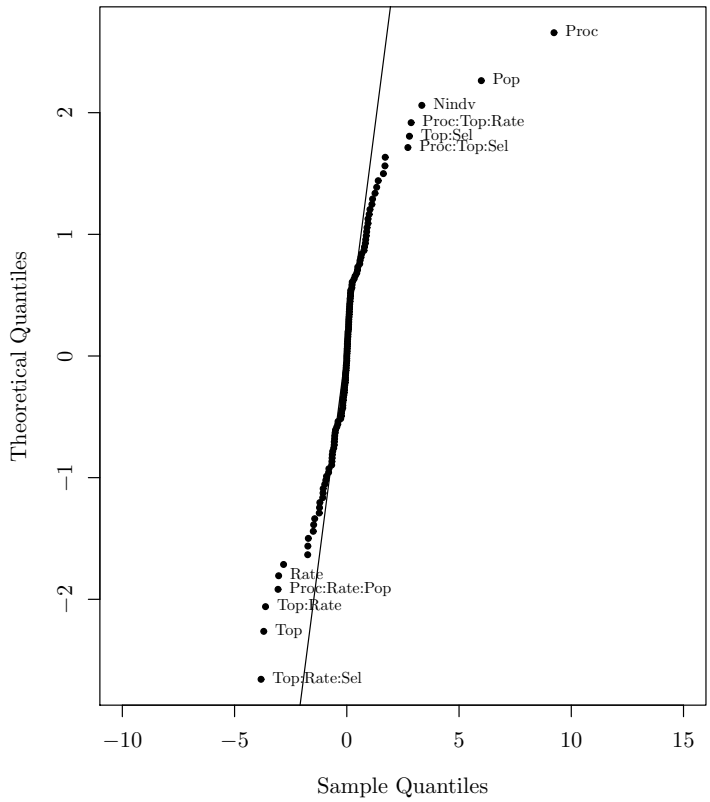


Figura 7.40: *Gráfico de probabilidade normal dos efeitos estimados do fatorial* 2^7 *dos trimmed* s_p *do AGP-I para a função Rastrigin no processador multicore Xeon.*

O efeito do fator Rep e de suas interações não foram significativos, e Rep foi removido. Assim, o fatorial 2^7 sem réplicas pode ser projetado em um fatorial 2^6 com uma réplica pelo método chamado *design projection* (Montgomery, 2009).

O passo seguinte da análise do fatorial foi executar o procedimento automático de seleção do modelo seguindo o critério AIC (Seção 4.4.1), através da função `stepAIC` do pacote MASS do R (Venables e Ripley, 2002).

Na Seção 7.4.1 foram definidos os níveis de significância adotados nos experimentos. Para a significância estatística foi considerado o nível de 5% de significância e efeitos maiores que 10% do *speedup* médio foram considerados ter significância prática. O modelo com os fatores e interações com significância estatística e prática é descrito na Tabela 7.17 e na forma da seguinte equação:

$$\widehat{y} = 2,318 - 0,816x_1 + 0,327x_2 - 0,269x_3 + 0,529x_4 + 0,296x_5 + 0,320x_2x_3, \tag{7.6}$$

onde as variáveis codificadas x_1, x_2, x_3, x_4 e x_5 representam os fatores Proc, Top, Rate, Pop e Nindv, respectivamente. O modelo apresentou o coeficiente de determinação R^2 igual a 0,593, o que significa que o modelo explica 59% da variação de $\widehat{y}$. O R^2 ajustado

foi de 0,573.

Tabela 7.17: *Modelo ajustado AGP-I para Rastrigin com trimmed mean.*

	Coeficiente	Erro Padrão	Valor t	Valor-p
(Interseção)	2,3168	0,0864	26,81	0,0000
Proc	-0,8158	0,0864	-9,44	0,0000
Pop	0,5295	0,0864	6,13	0,0000
Top	0,3271	0,0864	3,78	0,0002
Nindv	0,2954	0,0864	3,42	0,0009
Rate	-0,2684	0,0864	-3,11	0,0024
Top:Rate	0,3201	0,0864	3,70	0,0003

A verificação do modelo (7.6) através da análise dos resíduos é exibida na Figura 7.41. Do lado esquerdo, encontra-se o gráfico dos resíduos pelos valores preditos, e do lado direito, está o gráfico de probabilidade normal dos resíduos, o qual exibe os pontos posicionados próximos da reta. Os testes formais confirmaram que os resíduos estavam distribuídos de forma homogênea (teste *Breusch-Pagan*, função `ncvTest()` com valor-p = 0,059) e se aproximaram da distribuição normal (teste de *Shapiro-Wilk*, função `shapiro.test()` com valor-p = 0,122). Portanto, o modelo final (7.6) foi considerado válido.

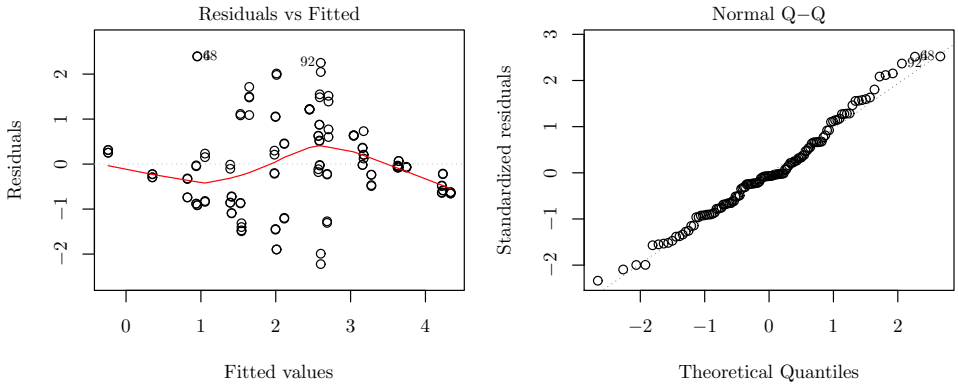


Figura 7.41: *Verificação dos resíduos do modelo (7.6): gráfico dos resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos.*

Os intervalos de confiança para os efeitos estimados para a variável y, o *trimmed* s_p, são expostos na Tabela 7.18. O diagrama de barra com os efeitos com significância estatística e prática são mostrados na Figura 7.42.

De acordo com o modelo para o *trimmed* s_p (7.6), os fatores Proc, Pop, Top, Nindv e Rate tiveram o efeito principal significativo. A seguir são interpretados os efeitos significativos do modelo (7.6):

- Proc: mudanças de quatro para 16 ilhas reduziam o *trimmed* s_p médio em 1,63.

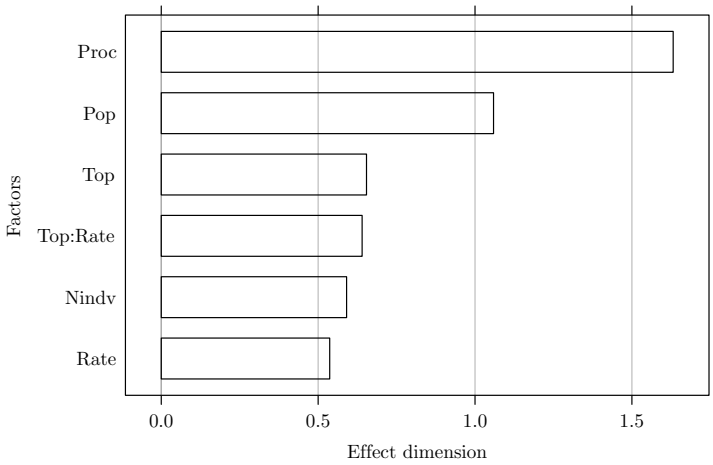


Figura 7.42: *Gráfico de barras com os efeitos estimados no trimmed s_p do AGP-I para função Rastrigin no processador multicore Xeon.*

Tabela 7.18: *Intervalo de confiança para os coeficientes estimados do modelo (7.6).*

Fator	Coeficiente	2.5 %	97.5 %
(Interseção)	2,32	2,15	2,49
Proc	-0,82	-0,99	-0,64
Pop	0,53	0,36	0,70
Top	0,33	0,16	0,50
Nindv	0,30	0,12	0,47
Rate	-0,27	-0,44	-0,10
Top:Rate	0,32	0,15	0,49

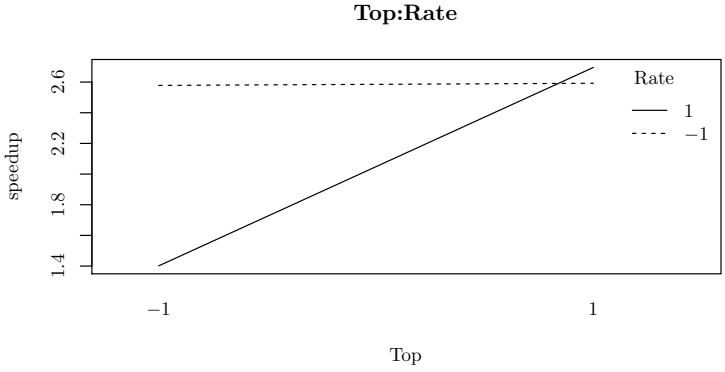


Figura 7.43: *Gráfico da interação Top:Rate do modelo (7.6).*

- Pop: quando o tamanho da população é alterado de 1600 para 3200, o *trimmed* s_p médio aumenta 1,06.

- Nindv: se o número de migrantes aumenta de 1 para 5, o *trimmed* s_p médio aumenta 0,59.

- *Top:Rate*: quando a taxa de migração é a cada 10 gerações, modificar a topologia de anel para totalmente ligada quase não altera o *trimmed* s_p, mas se a taxa de migração for a cada 100 gerações, a variação na topologia de migração de anel para totalmente ligada causa um aumento de 0,64 no *trimmed* s_p, exibido (Figura 7.43).

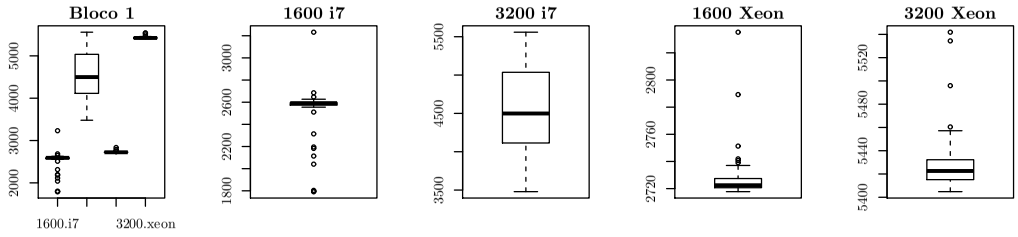
O aumento do tamanho da amostra nos testes executados no processador i7 não foi possível. No processador i7 foi aplicado um dos princípios básicos do planejamento experimental: a blocagem. Na próxima seção, o experimento que faz uso da blocagem no Xeon e no i7 é descrito.

7.10.5 Análise da Blocagem da Semente

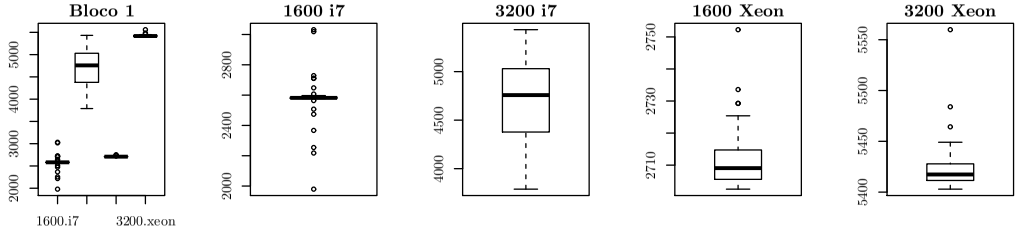
Um dos princípios básicos do planejamento experimental é a blocagem. A blocagem é uma técnica utilizada com o objetivo de aumentar a precisão de um experimento através do controle e avaliação de fatores, cuja influência na variabilidade da resposta não é de interesse do experimento. A semente do PRNG é reconhecida como fator que exerce influência no tempo de execução do AGP-I, porém não há interesse em estudá-la. A blocagem da semente foi utilizada na avaliação do *speedup* do AGP-I para função Rastrigin.

Foram escolhidas aleatoriamente seis sementes. Para cada semente foi estabelecido um bloco com 41 réplicas do experimento fatorial 2^k do AGP-I para a função Rastrigin. A Tabela 7.19 apresenta as estatísticas descritivas dos tempos de execução sequencial do AGPI para a função Rastrigin, separadas por bloco e população. Os valores de tempo sequencial obtidos para o bloco 6 com população de 1600 indivíduos foram muito maiores do que os dos outros blocos. A Figura 7.44 apresenta os diagramas de caixa dos tempos de execução sequencial para os blocos de 1 a 6.

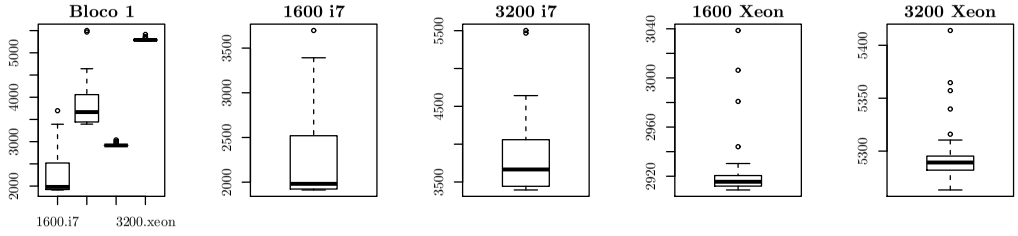
Os diagramas de caixa apresentados pela Figura 7.45 confirmam a discrepância dos tempos apresentados pelo bloco 6 em relação aos outros blocos, especificamente para populações de 1600 indivíduos.



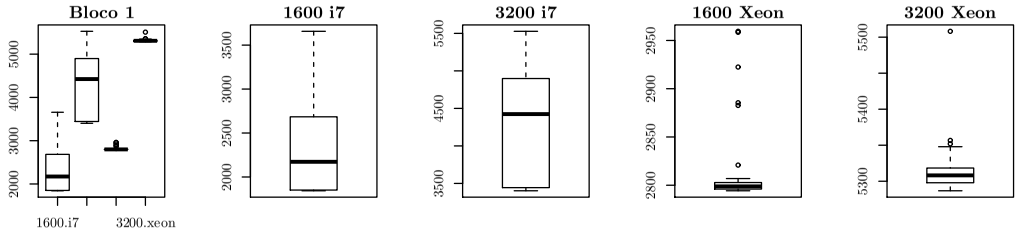
(a) Bloco 1



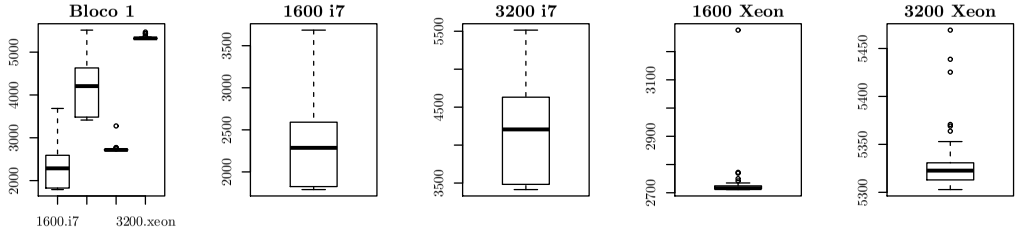
(b) Bloco 2



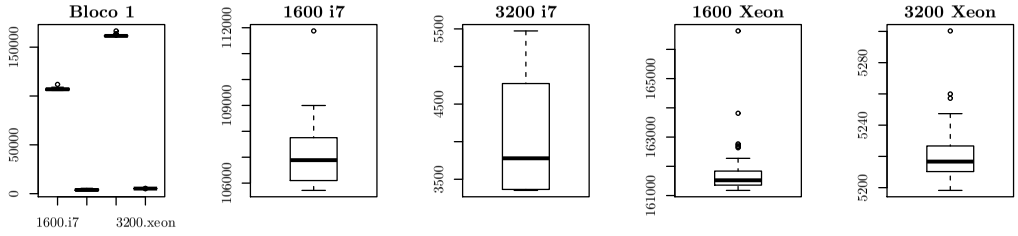
(c) Bloco 3



(d) Bloco 4



(e) Bloco 5



(f) Bloco 6

Figura 7.44: *Diagramas de caixa para os tempos de execução sequencial (ms) do AGP-I para função Rastrigin por blocos.*

Tabela 7.19: *Estatísticas tempos de execução sequencial nos seis blocos, $n = 41$ em cada bloco.*

	grupo	$\overline{X}$	S	mediana	min	max	assim.	curtose	SE	δ_X	$\delta_{\overline{X}}$
B1	Xeon 1600	2729,52	21,09	2722,48	2717,76	2835,21	3,61	14,00	3,29	0,01	0,00
B1	Xeon 3200	5431,41	29,75	5422,58	5404,81	5541,88	2,46	5,79	4,65	0,01	0,00
B1	i7 1600	2499,89	270,05	2591,53	1792,39	3232,11	-1,04	2,10	42,17	0,11	0,02
B1	i7 3200	4517,77	540,81	4498,74	3479,26	5559,16	-0,07	-0,96	84,46	0,12	0,02
B2	Xeon 1600	2712,18	10,03	2709,06	2702,51	2752,26	1,99	4,39	1,57	0,00	0,00
B2	Xeon 3200	5425,64	27,71	5417,35	5402,91	5559,89	3,01	11,12	4,33	0,01	0,00
B2	i7 1600	2574,29	167,16	2581,72	1979,51	3031,79	-0,55	4,43	26,11	0,06	0,01
B2	i7 3200	4676,78	424,67	4758,08	3789,05	5432,19	-0,46	-0,93	66,32	0,09	0,01
B3	Xeon 1600	2923,60	25,96	2915,50	2908,82	3038,62	3,13	9,52	4,05	0,01	0,00
B3	Xeon 3200	5296,00	27,82	5289,23	5263,28	5413,99	2,46	6,63	4,35	0,01	0,00
B3	i7 1600	2255,08	472,66	1981,50	1912,79	3697,07	1,32	0,79	73,82	0,21	0,03
B3	i7 3200	3853,52	523,80	3665,59	3394,78	5502,41	1,53	2,10	81,80	0,14	0,02
B4	Xeon 1600	2814,17	42,52	2798,82	2794,25	2959,49	2,50	4,97	6,64	0,02	0,00
B4	Xeon 3200	5315,96	34,94	5308,13	5286,76	5508,13	4,08	19,46	5,46	0,01	0,00
B4	i7 1600	2334,23	544,45	2172,70	1843,14	3657,01	0,80	-0,53	85,03	0,23	0,04
B4	i7 3200	4307,80	772,58	4423,92	3402,14	5527,57	0,17	-1,51	120,66	0,18	0,03
B5	Xeon 1600	2735,18	87,73	2716,17	2711,24	3276,11	5,71	32,25	13,70	0,03	0,01
B5	Xeon 3200	5333,10	35,71	5322,62	5302,85	5469,15	2,38	5,22	5,58	0,01	0,00
B5	i7 1600	2295,64	474,71	2286,22	1790,16	3685,31	0,74	-0,04	74,14	0,21	0,03
B5	i7 3200	4223,11	701,61	4206,44	3414,03	5514,68	0,40	-1,15	109,57	0,17	0,03
B6	Xeon 1600	161802,87	951,76	161521,98	161176,12	166630,93	3,43	13,76	148,64	0,01	0,00
B6	Xeon 3200	5221,13	19,54	5216,69	5198,24	5300,31	1,85	4,62	3,05	0,00	0,00
B6	i7 1600	107074,64	1205,38	106886,12	105718,68	111876,36	1,55	3,83	188,25	0,01	0,00
B6	i7 3200	4061,85	787,26	3778,79	3351,99	5473,18	0,73	-1,12	122,95	0,19	0,03

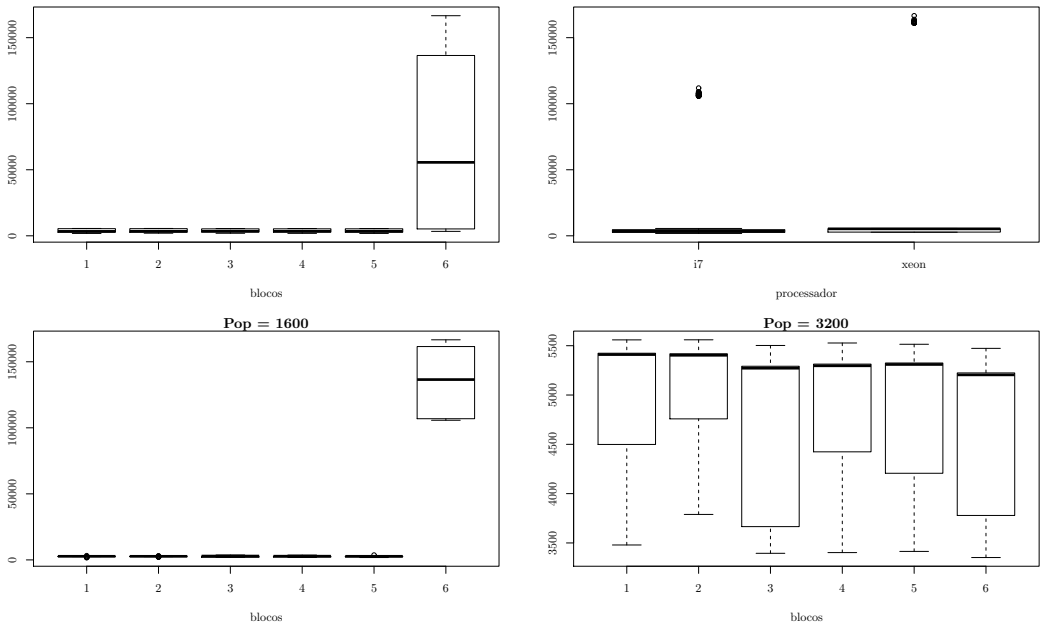


Figura 7.45: *Tempos sequenciais (ms) da execução do AGP-I para a função Rastrigin exibidos em diagramas de caixa. Na parte superior, da esquerda para a direita, agrupados por bloco e agrupados por processador. Na parte inferior, da esquerda para a direita, para população com 1600 e agrupados por bloco e para população com 3200 agrupado por bloco.*

A Figura 7.46 apresenta os tempos de execução sequencial (ms) separados por tamanho da população e processador *multicore*. Os diagramas de caixa da coluna à esquerda na Figura 7.46 confirmam que o bloco 6 é discrepante dos demais blocos quando a população é de 1600, e que essa discrepância ocorre independente do processador *multicore*. As Figuras 7.47 e 7.48 exibem os diagramas de caixa para os cinco blocos restantes

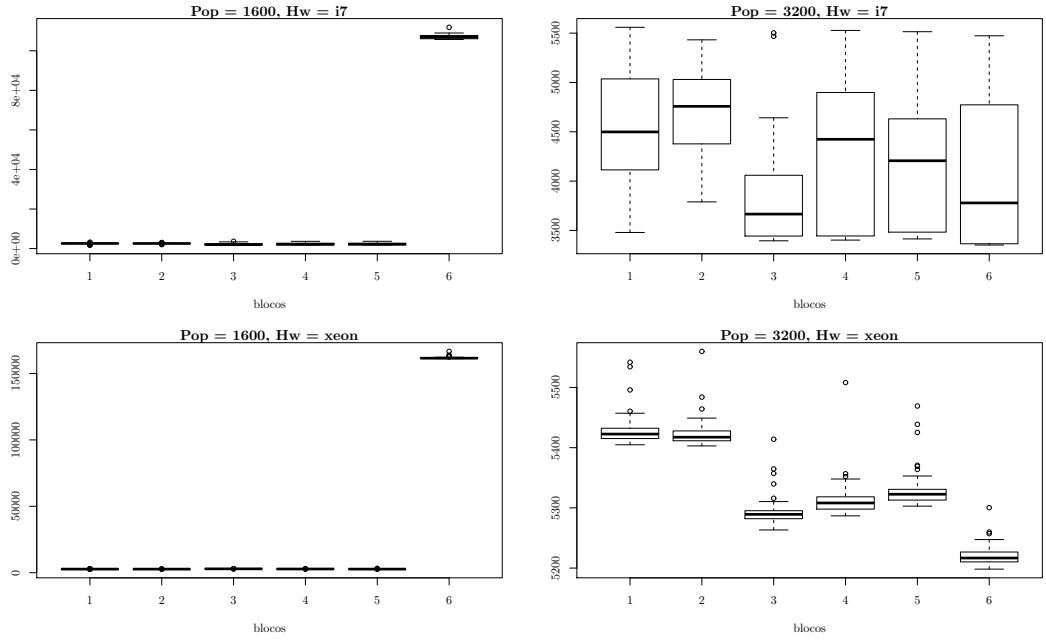


Figura 7.46: *Tempos sequenciais (ms) da execução do AGP-I para a função Rastrigin exibidos em diagramas de caixa, separados por tamanho da população e processador multicore.*

após a remoção do bloco 6.

Speedups

A análise dos tempos paralelos verificou primeiro as condições estabelecidas por Qiao et al. (2006) e Díaz-Francés e Rubio (2013) (ver Seção 6.6.2) em todos os blocos. As condições foram satisfeitas. Assim, foi garantida que a razão das médias $\overline{X}/\overline{Y}$ do *speedup* tenha uma distribuição aproximada da distribuição normal. Foram encontrados casos em que δ_{Y_i} estava acima do limite de $0,2$, porém todos os $\delta_{\overline{Y_i}}$ estavam abaixo do limite de $0,2$. Desta forma, o *speedup* em cada bloco foi calculado como uma razão ponderada (Equação 6.10). As estatísticas descritivas dos *speedups* obtidos nos blocos são apresentadas na Tabela 7.20, onde a primeira linha corresponde a todos os *speedups* sem distinção de bloco, e nas linhas seguintes os *speedup* estão separados por bloco.

A Figura 7.49 apresenta o diagrama de caixa dos *speedups* por bloco, o qual mostra que a variação e a distribuição dos *speedups* é semelhante nos diferentes blocos.

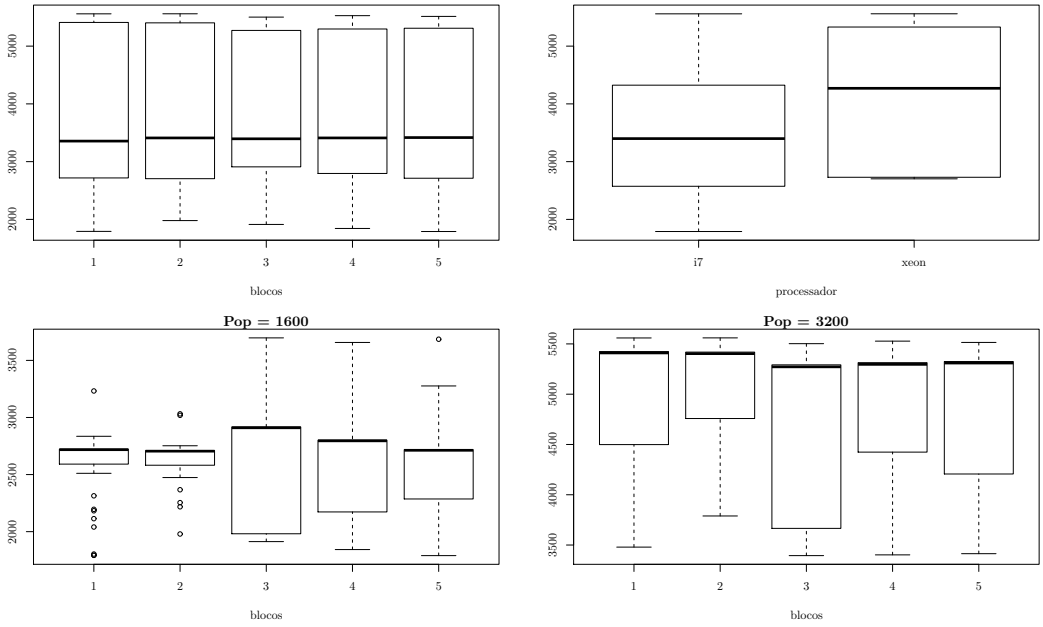


Figura 7.47: *Tempos sequenciais (ms) da execução do AGP-I para a função Rastrigin exibidos em diagramas de caixa após a retirada do bloco 6. Na parte superior, da esquerda para a direita, agrupados por bloco e agrupados por processador. Na parte inferior, da esquerda para a direita, para população com 1600 e agrupados por bloco e para população com 3200 agrupado por bloco.*

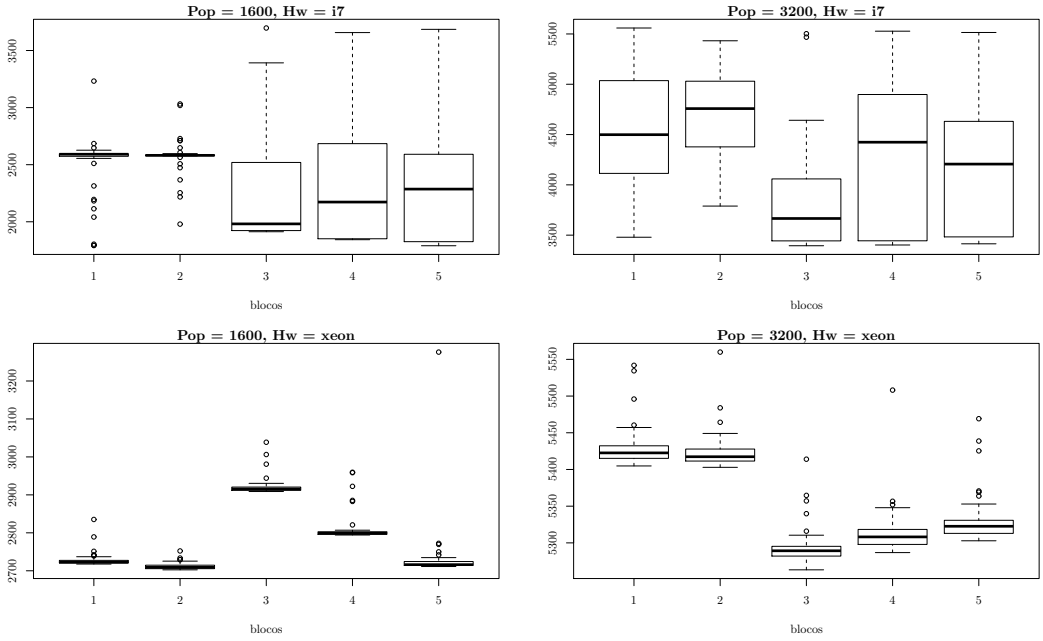


Figura 7.48: *Tempos sequenciais (ms) da execução do AGP-I para a função Rastrigin exibidos em diagramas de caixa, separados por tamanho da população e processador multicore, após a remoção do bloco 6.*

Tabela 7.20: *Estatísticas descritivas dos speedups s_p do AGP-I para função Rastrigin com blocagem da semente.*

	média $\overline{s_p}$	S	mediana	min	max	intervalo	assim.	curtose	SE
Todos	2,17	1,39	2,52	0,01	6,42	6,41	-0,21	-0,99	0,04
Bloco 1	2,11	1,39	2,52	0,01	6,42	6,41	-0,11	-1,03	0,09
Bloco 2	2,12	1,38	2,52	0,01	5,88	5,87	-0,16	-1,06	0,09
Bloco 3	2.23	1.42	2.54	0.01	5.96	5.95	-0.35	-1.14	0.09
Bloco 4	2.25	1.39	2.57	0.01	6.05	6.04	-0.29	-0.89	0.09
Bloco 5	2.16	1.37	2.49	0.01	6.36	6.35	-0.15	-0.80	0.09

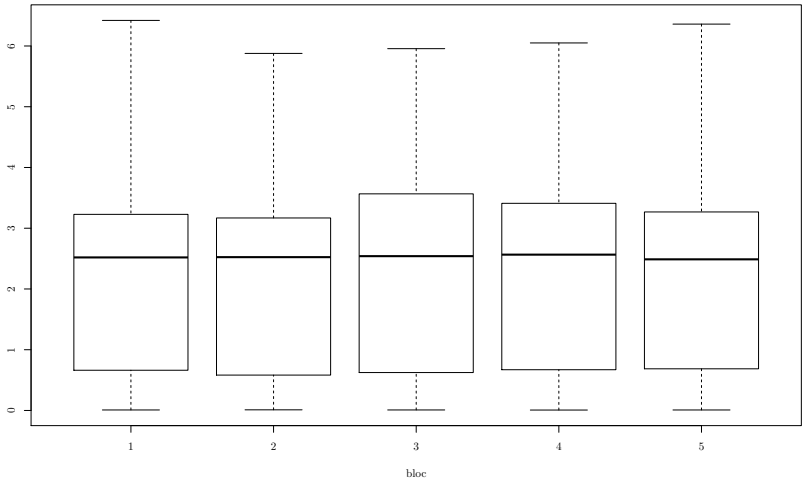


Figura 7.49: *Diagramas de caixa dos speedups s_p agrupados por bloco.*

A Figura 7.50 exibe os diagramas de caixa dos *speedups* por fator. O primeiro diagrama de caixa na linha superior, à esquerda, mostra que, para o fator Proc, a metade central dos *speedups* para quatro ilhas é superior à dos *speedups* para 16 ilhas. O digrama de caixa para o fator Rep exibe duas caixas quase idênticas para os diferentes níveis, indicando que a mudança nos seus níveis não influi no *speedup*.

Os gráficos exploratórios dos *speedups* s_p são apresentados na Figura 7.51, e a sua classificação como: sublinear, linear e superlinear, é exibida na Figura 7.52, onde os símbolos correspondem à classificação dos *speedups* (ver Seção 6.5.3).

A análise de variância para o planejamento com blocos completos é exibida na Tabela 7.21. O efeito dos blocos não foi significativo e os *speedups* foram analisados como um planejamento fatorial 2^8 com cinco réplicas.

O cálculo do efeito dos fatores sobre o *speedup* foi realizado através do modelo de regressão linear. O modelo completo do fatorial 2^k foi ajustado automaticamente pela função `stepAIC` do pacote MASS (Venables e Ripley, 2002) do R. O modelo ajustado apresentado na Tabela 7.22 incluiu os efeitos estatisticamente significativos.

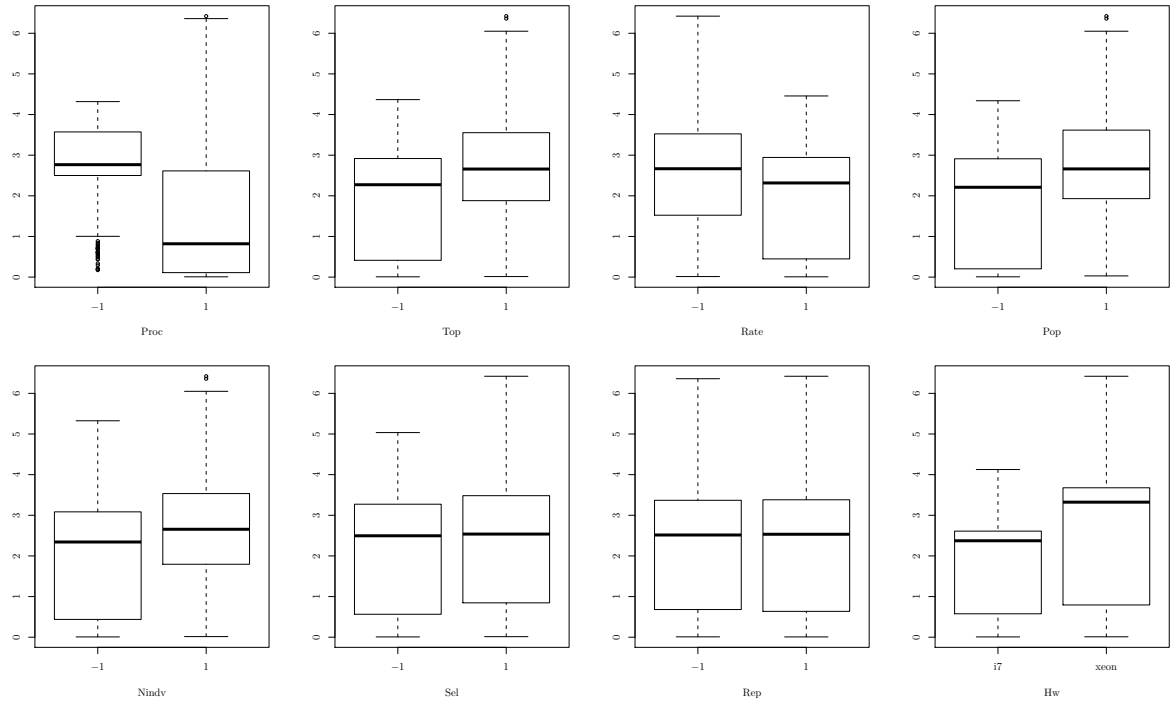


Figura 7.50: *Diagramas de caixa dos speedups s_p agrupados por fator.*

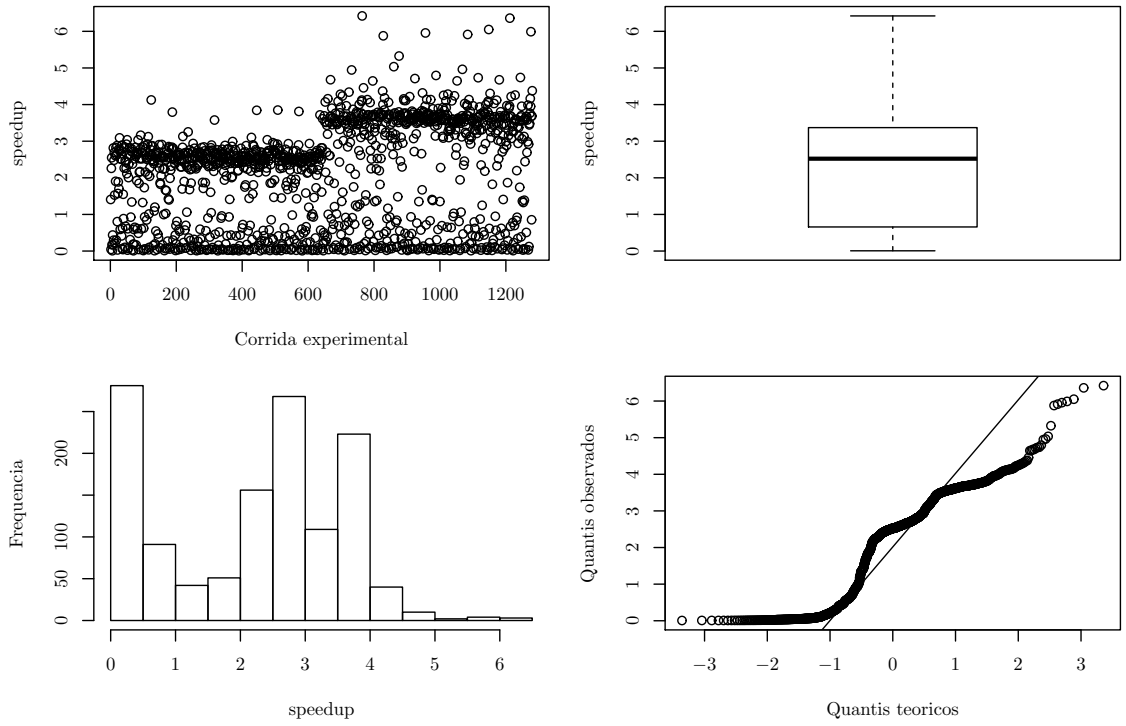


Figura 7.51: *Gráficos exploratórios dos speedups do AGP-I para função Rastrigin, com blocagem da semente.*

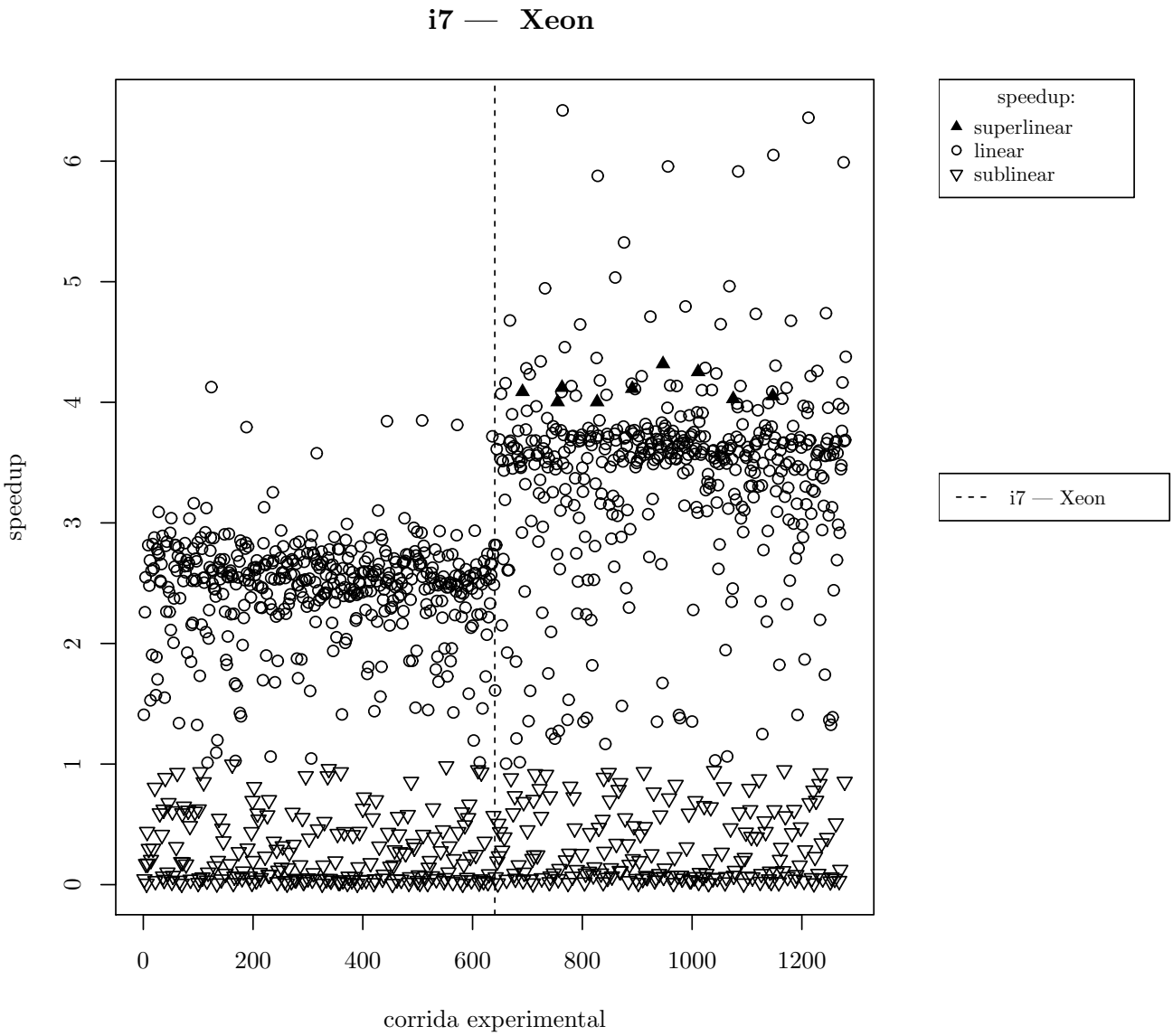


Figura 7.52: *Gráfico dos speedups s_p por corrida experimental, com blocagem da semente. A linha vertical tracejada separa os s_p por processador multicore.*

Tabela 7.21: *Análise de variância para o planejamento com blocos completos.*

Fonte de Variação	Graus de Liberdade	Soma dos Quadrados	Média Quadrática	F	Valor-p
Proc	1	621,08	621,08	687,98	0,0000
Top	1	139,75	139,75	154,81	0,0000
Rate	1	85,86	85,86	95,11	0,0000
Pop	1	190,47	190,47	210,98	0,0000
Nindv	1	95,40	95,40	105,68	0,0000
Sel	1	5,30	5,30	5,87	0,0155
Rep	1	0,06	0,06	0,06	0,8018
Hw	1	184,36	184,36	204,21	0,0000
bloc	4	4,01	1,00	1,11	0,3501
Resíduos	1267	1143,80	0,90		

O coeficiente de determinação R^2 do modelo foi 0,692, com 1251 graus de liberdade, ou seja, o modelo explica 69,2% da variação em $\widehat{y^*}$.

Tabela 7.22: *Modelo ajustado para o planejamento fatorial* 2^8 *com blocagem.*

	Coeficiente	Erro Padrão	Valor t	Valor-p
(Interseção)	2,1730	0,0218	99,70	0,0000
Proc	-0,6966	0,0218	-31,96	0,0000
Pop	0,3857	0,0218	17,70	0,0000
Hw	-0,3795	0,0218	-17,41	0,0000
Top	0,3304	0,0218	15,16	0,0000
Nindv	0,2730	0,0218	12,53	0,0000
Rate	-0,2590	0,0218	-11,88	0,0000
Sel	0,0644	0,0218	2,95	0,0032
Proc:Top	0,2379	0,0218	10,91	0,0000
Top:Rate	0,1705	0,0218	7,82	0,0000
Proc:Nindv	0,1654	0,0218	7,59	0,0000
Proc:Pop	0,1340	0,0218	6,15	0,0000
Pop:Hw	-0,1048	0,0218	-4,81	0,0000
Proc:Rate	-0,0782	0,0218	-3,59	0,0003
Hw:Top	-0,0741	0,0218	-3,40	0,0007
Proc:Hw	0,0709	0,0218	3,25	0,0012
Top:Sel	-0,1626	0,0218	-7,46	0,0000
Rate:Sel	0,0816	0,0218	3,74	0,0002
Hw:Nindv	-0,0556	0,0218	-2,55	0,0109
Top:Nindv	0,0505	0,0218	2,32	0,0206
Hw:Rate	0,0494	0,0218	2,27	0,0235
Pop:Sel	0,0449	0,0218	2,06	0,0395
Proc:Top:Rate	0,1399	0,0218	6,42	0,0000
Top:Rate:Sel	0,2342	0,0218	10,75	0,0000
Proc:Hw:Top	-0,0521	0,0218	-2,39	0,0171
Proc:Top:Nindv	0,0560	0,0218	2,57	0,0103
Proc:Pop:Nindv	0,1044	0,0218	4,79	0,0000
Proc:Pop:Top	0,0862	0,0218	3,96	0,0001
Pop:Top:Nindv	0,0453	0,0218	2,08	0,0381

Na Seção 7.4.1 foram definidos os níveis de significância adotados nos experimentos. Para a significância estatística foi considerado o nível de 5% de significância, e efeitos maiores que 10% do *speedup* médio foram considerados ter significância prática. O modelo final, considerando-se os níveis de significância estatística e prática adotados, é exibido na Tabela 7.23, e os intervalos de confiança para os coeficientes estimados estão na Tabela 7.24. O gráfico com os valores absolutos do valor estimado de cada efeito no s_p médio do AGP-I para função Rastrigin é exibido na Figura 7.53.

O modelo final, com os coeficientes com significância estatística e prática, corresponde à seguinte equação:

$$\begin{aligned}\widehat{y} =& 2,173 - 0,697x_1 + 0,386x_4 - 0,380x_8 + 0,330x_2 + 0,273x_5 - 0,259x_3 + 0,238x_1x_2 \\ &+ 0,171x_2x_3 + 0,165x_1x_5 - 0,163x_2x_6 + 0,134x_1x_4 + 0,234x_2x_3x_6 + 0,140x_1x_2x_3\end{aligned} \tag{7.7}$$

onde as variáveis codificadas x_1, x_2, x_3, x_4, x_5, x_6 e x_8 representam os fatores Proc,

Tabela 7.23: *Modelo final do speedup do AGP-I para Rastringin com bloqueio da semente.*

	Coeficiente	Erro Padrão	Valor t	Valor-p
(Interseção)	2,1730	0,0230	94,53	0,0000
Proc	-0,6966	0,0230	-30,30	0,0000
Pop	0,3857	0,0230	16,78	0,0000
Hw	-0,3795	0,0230	-16,51	0,0000
Top	0,3304	0,0230	14,37	0,0000
Nindv	0,2730	0,0230	11,88	0,0000
Rate	-0,2590	0,0230	-11,27	0,0000
Proc:Top	0,2379	0,0230	10,35	0,0000
Top:Rate	0,1705	0,0230	7,42	0,0000
Proc:Nindv	0,1654	0,0230	7,19	0,0000
Top:Sel	-0,1626	0,0230	-7,07	0,0000
Proc:Pop	0,1340	0,0230	5,83	0,0000
Top:Rate:Sel	0,2342	0,0230	10,19	0,0000
Proc:Top:Rate	0,1399	0,0230	6,09	0,0000

Tabela 7.24: *Intervalo de confiança para o modelo final (7.7).*

	Coeficiente	2,5 %	97,5 %
(Interseção)	2,17	2,13	2,22
Proc	-0,70	-0,74	-0,65
Pop	0,39	0,34	0,43
Hw	-0,38	-0,42	-0,33
Top	0,33	0,29	0,38
Nindv	0,27	0,23	0,32
Rate	-0,26	-0,30	-0,21
Proc:Top	0,24	0,19	0,28
Top:Rate	0,17	0,13	0,22
Proc:Nindv	0,17	0,12	0,21
Top:Sel	-0,16	-0,21	-0,12
Proc:Pop	0,13	0,09	0,18
Top:Rate:Sel	0,23	0,19	0,28
Proc:Top:Rate	0,14	0,09	0,19

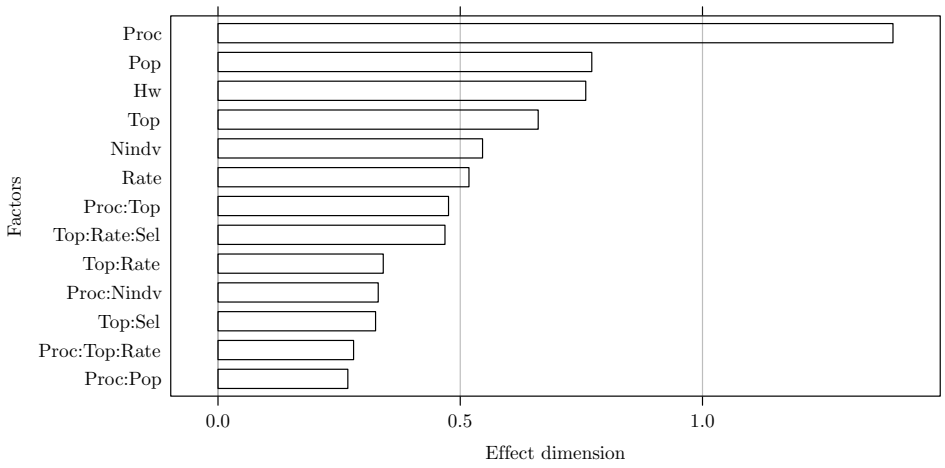


Figura 7.53: *Diagrama de barra com os valores absolutos do valor estimado de efeito no s_p médio do AGP-I para a função Rastrigin.*

Top, Rate, Pop, Nindv, Sel e Hw, respectivamente. O modelo apresentou o coeficiente de determinação R^2 igual a 0.653, o que significa que o modelo explica 65% da variação de $\widehat{y}$. O R^2 ajustado foi 0.649.

A verificação do ajuste do modelo através da análise dos resíduos foi realizada. A Figura 7.54 apresenta os gráficos dos resíduos para verificação dos pressupostos de variância constante e de proximidade com a distribuição normal. A variabilidade dos resíduos nos níveis de cada fator e nos blocos é exibida nas Figuras 7.55 e 7.56. O gráfico dos resíduos pelos valores preditos sugere que a variância não é constante, apesar do teste formal, como é mostrado a seguir, indicar o contrário. A desigualdade de variância sinaliza que o modelo pode ser melhorado através da adição de variáveis preditoras, e/ou que o modelo não é linear.

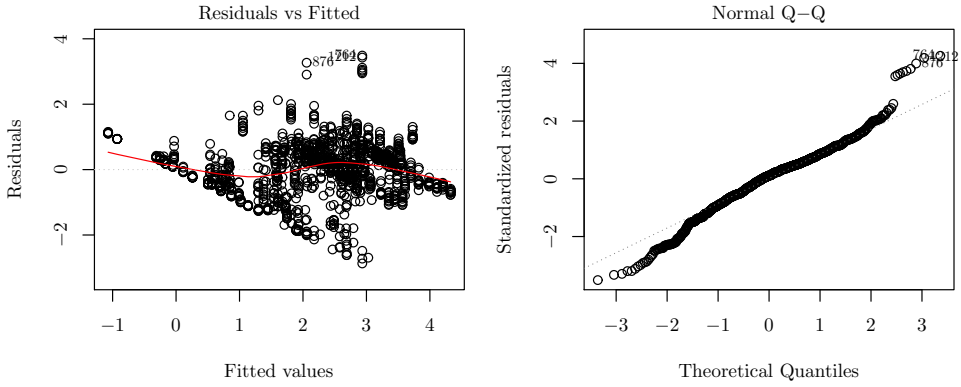


Figura 7.54: *Verificação dos resíduos do modelo (7.7).*

Os testes formais indicaram que os resíduos estavam distribuídos de forma homogênea (teste *Breusch-Pagan*, função `ncvTest()` com valor-p = 0,44), porém não se aproximaram da distribuição normal (teste de *Shapiro-Wilk*, função `shapiro.test()` com valor-p = $6{,}68.10^{-14}$). De acordo com Montgomery (2009), desvios de normalidade moderados podem ocorrer contanto que pontos discrepantes sejam investigados. Testes formais podem auxiliar na avaliação dos discrepantes, como a distância de Cook, os resíduos *studentized*, o método de ajuste de Bonferroni (Kutner et al., 2005).

Para o modelo final (7.7), o histograma dos resíduos *studentizados* e a estimativa da sua função densidade são exibidos na Figura 7.57. O resíduos *studentizados* apresentam uma distribuição simétrica e muito próxima da distribuição normal.

As funções `outlierTest` e `influencePlot`, do pacote `car` (Fox e Weisberg, 2011) do R, foram executadas. Os pontos 764 e 1212 foram identificados como discrepantes, com resíduos *studentized* de 4,29 e 4,22, valor-p ajustados pelo método de Bonferroni

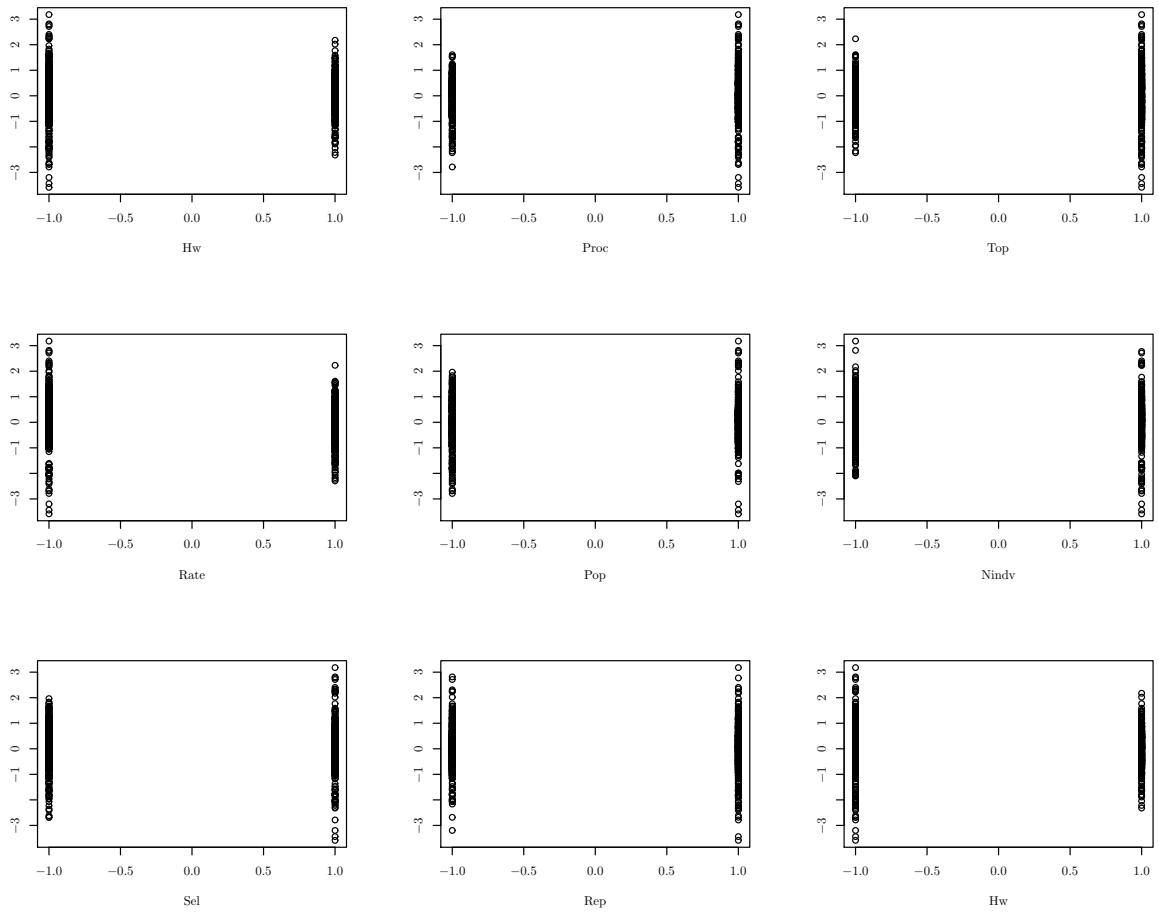


Figura 7.55: *Gráficos dos resíduos do modelo (7.7) pelos níveis dos fatores.*

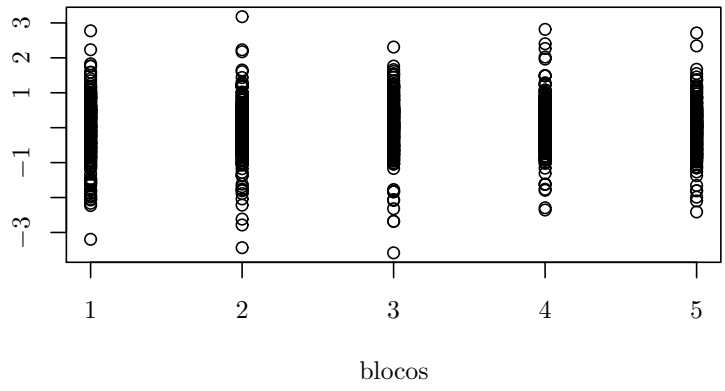


Figura 7.56: *Gráficos dos resíduos do modelo (7.7) pelos blocos.*

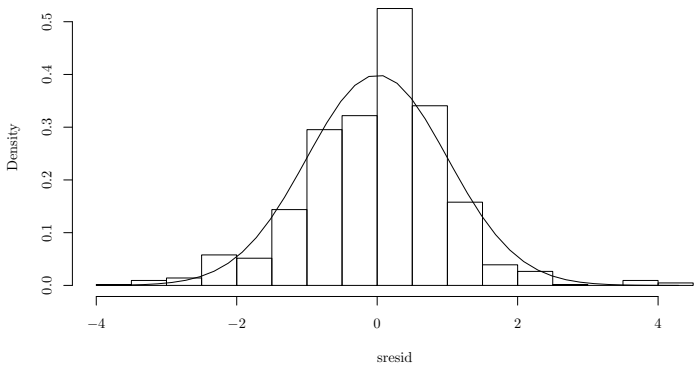


Figura 7.57: *Histograma dos resíduos studentized do modelo (7.7).*

igual a 0,024 e 0,034, e distância de Cook com valor 0,1198 e 0,1177. Fox (1991) sugere que o valor da distância de Cook seja usada de forma conjunta com os valores dos outros pontos, sem se ater a um valor limite para definir pontos discrepantes. O autor aconselha investigar os pontos que são notadamente maiores do que a maioria dos outros pontos. A Figura 7.58 mostra as distâncias de Cook para os pontos observados. Os pontos 764, 1212 não diferem substancialmente dos outros pontos do grupo de observação do i7. Portanto, de modo geral o problema de desvio de normalidade não é severo o suficiente para ter grande impacto na análise e nas conclusões. O modelo final (7.7) foi considerado válido.

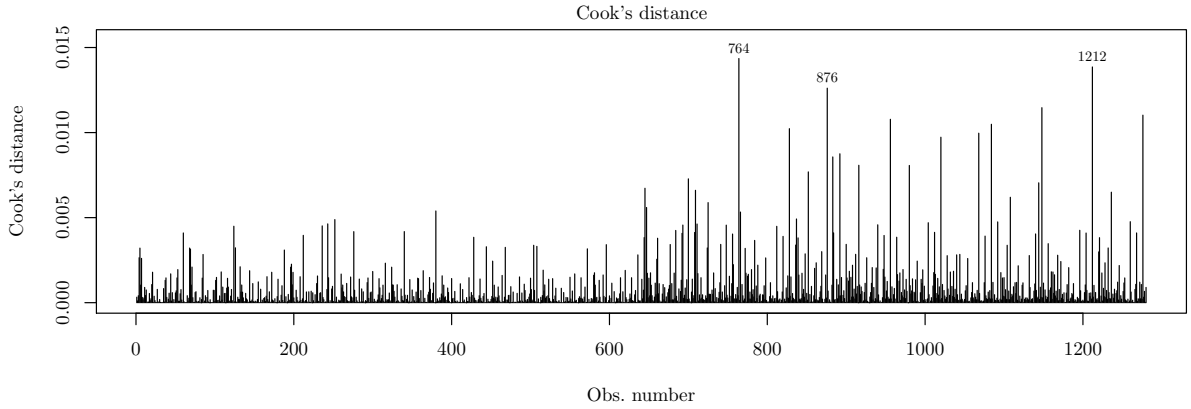


Figura 7.58: *Diagrama de com as distâncias de Cook para os os pontos observados segundo o modelo final (7.7).*

Os efeitos principais significativos foram os efeitos dos fatores Proc, Pop, Hw, Top, Nindv e Rate. Todos esses fatores, com exceção do fator Hw, participaram de interações e não podem ser interpretados sozinhos, mas junto com os outros fatores da interação.

- Duas interações de terceira ordem foram significativas: Top:Rate:Sel e Proc:Top:Rate, com efeitos estimados em 0,4684 e 0,2798.

- A interação de terceira ordem Top:Rate:Sel foi significativa, assim como as interações de segunda ordem Top:Rate e Top:Sel. A Figura 7.59 ajuda a interpretar a interação de terceira ordem: dependendo do nível em que um dos fatores está, a interação de segunda ordem dos outros dois fatores terá valores diferentes. A Figura 7.59 mostra que o gráfico da interação Top:Rate é diferente nos dois níveis do fator Sel.

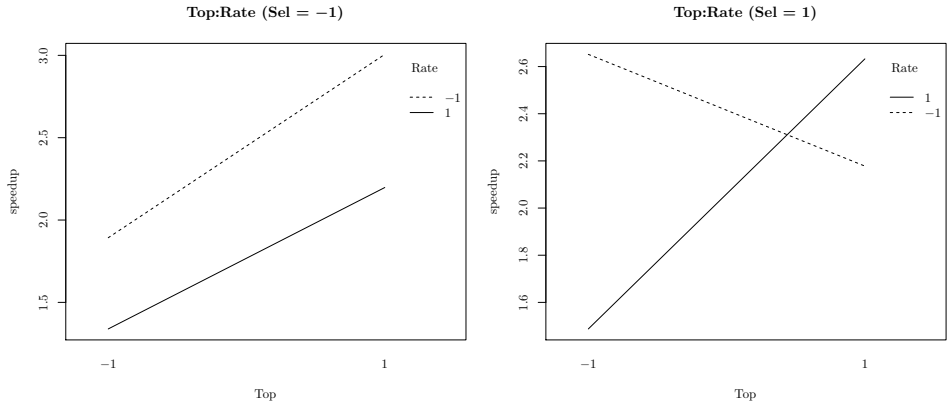


Figura 7.59: *Gráfico da interação Top:Rate:Sel.*

- A interação de terceira ordem Proc:Top:Rate foi significativa e com efeito estimado em 0,2798. As interações de segunda ordem, Proc:Top e Top:Rate, que estão envolvidas nessa interação de terceira ordem, tem os efeitos estimados em 0,4758 e 0,3410. A Figura 7.60 mostra que para cada nível do fator Proc, os gráficos da interação Top:Rate são diferentes.

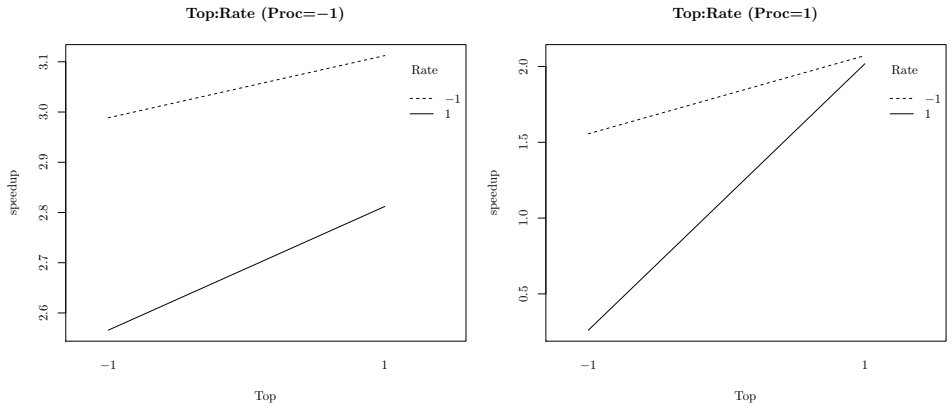


Figura 7.60: *Gráfico da interação Proc:Top:Rate.*

- As interações de segunda ordem, Proc:Nindv e Proc:Pop com valor 0,3308 e 0,2680, são significativas e não estão em interações de terceira ordem significativas. Os seus gráficos de interação são exibidos na Figura 7.61.

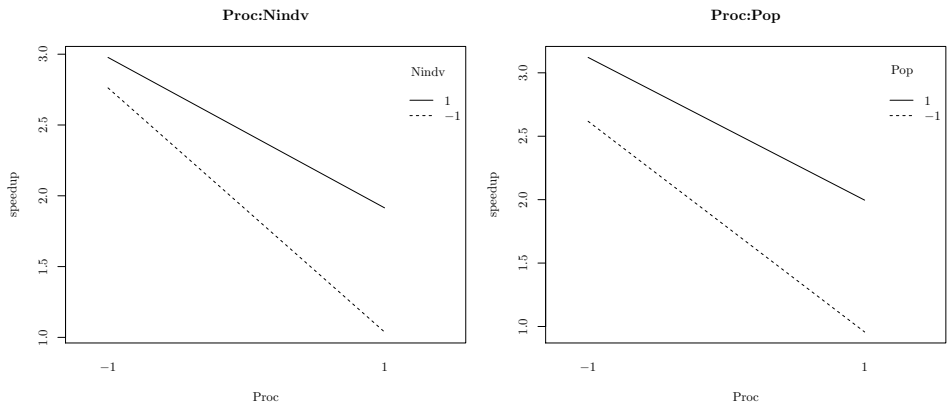


Figura 7.61: *Gráfico das interações com dois fatores: Proc:Nindv e Proc:Pop.*

- O efeito principal Hw foi significativo e negativo. Uma mudança do fator Hw de Xeon para i7 representou a diminuição do *speedup* de -0,7590.

7.10.6 Discussão

Assim como para a função Rosenbrock, para a função Rastrigin, os tempos sequenciais foram observados em quatro grupos diferentes, definidos pelo processador *multicore* e o tamanho da população, que são Xeon com 1600, Xeon com 3200, i7 com 1600 e i7 com 3200.

Foram identificados dois pontos discrepantes extremos no grupos com 1600 indivíduos, os pontos com rótulos 1311 e 261. O estudo desses pontos extremos revelou que foram fruto da influência da semente do PRNG. A sequência de números pseudo-aleatórios determinou que fossem necessárias quantidades de gerações extremas para se chegar a uma solução. A eliminação desses pontos extremos fez com que outros pontos discrepantes não tão extremos ficassem em evidência.

A aplicação do método *trimmed mean* para eliminar os pontos discrepantes não sucedeu quanto às condições para que a distribuição da razão entre as médias dos tempos sequenciais e paralelos se aproximasse da distribuição normal.

O tamanho da amostra foi aumentado, porém apenas no processador Xeon. O processador i7 não estava disponível para os testes. Mesmo assim, os tempos sequenciais para os grupos com 1600 indivíduos apresentou valores discrepantes e δ_{X_i} que não satisfez as condições do Teorema 6.6.1.

O método do *trimmed mean* foi aplicado e as condições de aproximação com a distribuição da razão entre a média dos tempos sequenciais e paralelos foram satisfeitas. Os *speedups* foram calculados e o planejamento fatorial 2^7 foi analisado.

Os tempos observados para a função Rastrigin após a aplicação do método *trimmed mean* foram menores do que seis segundos. Tempos menores do que os da função Rosenbrock. Proporcionalmente, as variações causadas pelos fatores não controlados têm um impacto maior nos tempos de execução menores, causando maior variabilidade nos tempos observados para a função Rastrigin do que os observados para a função Rosenbrock.

Capítulo 8

Considerações Finais

Os AE têm demonstrado habilidade em tratar os mais variados problemas de otimização, fornecendo soluções viáveis e próximas da solução ótima. Outra característica vantajosa dos AE é o seu paralelismo intrínseco. Essa característica tem aumentado de importância no cenário atual, com a adoção dos processadores *multicore* no desenvolvimento dos computadores .

Os processadores *multicore* possuem diferentes arquiteturas, com particularidades em relação ao compartilhamento de recursos dentro do processador e às tecnologias implementadas para a otimização de desempenho, como o *Hyperthreading* e o *Turbo Boost*. A capacidade de processamento dos processadores *multicore* fundamenta-se no trabalho em paralelo dos seus núcleos. Essa característica favorece os programas paralelos e são um incentivo à paralelização dos AE, uma vez que os *multicore* estão largamente disponíveis.

A flexibilidade dos AE na resolução de diferentes tipos de problemas possui um custo associado: o esforço de ajustar os seus parâmetros para obter um melhor desempenho. Esse custo é agravado quando o AE é paralelizado e o número de parâmetros chega facilmente a duas casas decimais, com parâmetros qualitativos e quantitativos, que podem assumir inúmeros valores.

É necessária uma metodologia compreensiva que permita avaliar os AEP no tratamento de diferentes tipos de problemas, em diferentes plataformas *multicore*, de forma eficiente e com resultados válidos.

Para tanto, neste trabalho foram utilizadas as técnicas de planejamento estatístico experimental. O planejamento experimental representa um conjunto de ensaios estabe-

lecidos com critérios científicos e estatísticos, com o objetivo de determinar a influência de diversas variáveis nos resultados de um sistema ou processo. Um experimento bem planejado pode direcionar a modelagem do desempenho do AEP.

As etapas da metodologia de desenvolvimento deste trabalho estão definidas na seção 1.3. A primeira etapa, a revisão bibliográfica, está relatada nos Capítulos 2, 3, 4, 5 e parte do 6. A segunda etapa, de definição da metodologia de ajuste de parâmetros de AGPs em plataforma *multicore*, é descrita no Capítulo 6. Os estudos de caso estão relatados no Capítulo 7. Os Capítulos 2, 3 e 4 trazem os conceitos teóricos básicos para o desenvolvimento deste trabalho: Algoritmos Genéticos Paralelos, processadores *multicore* e planejamento e análise de experimentos.

O Capítulo 5 relata a revisão bibliográfica sobre o estudo da influência dos parâmetros de metaheurísticas. Diferentes abordagens para a solução desse problema são encontradas na literatura. São apresentados com mais profundidade os trabalhos que fazem uso de técnicas de planejamento experimental no ajuste de parâmetros de heurísticas, metaheurísticas e algoritmos da Computação Evolutiva.

O Capítulo 6 apresenta a metodologia de experimentação com AEP a partir de uma extensa revisão bibliográfica sobre a experimentação com algoritmos. A metodologia faz uso do planejamento fatorial 2^k na avaliação dos fatores que influenciam o desempenho dos AEP.

No Capítulo 7 são descritos experimentos realizados de acordo com a metodologia apresentada. Os experimentos realizados atestam a complexidade da experimentação com algoritmos.

Principais considerações em relação aos resultados apresentados, às limitações encontradas e trabalhos a serem executados futuramente:

- Os tempos de execução nos processadores *multicore* possuem uma variabilidade que dificulta a sua análise. Os fatores não-controláveis podem ser os processos executados em segundo plano, a concorrência pelos recursos compartilhados no processador e as tecnologias implementadas para a otimização do desempenho do processador.

- O processador *multicore* Intel i7 apresentou uma variabilidade maior do que o processador Xeon. O i7 implementa o *Hyperthreading* e o *Turbo Boost* que influenciam os tempos de execução total e são fatores não-controláveis.

- A análise dos resultados observados para a função Rastrigin foi mais complexa do que a da função Rosenbrock, devido a maior variabilidade dos tempos de execução observados. Essa maior variabilidade tem duas fontes: a característica multimodal da função Rastrigin e os seus tempos de execução pequenos. Por ser multimodal, certas sequências de números pseudo-aleatórios podem levar a um número muito maior de gerações necessárias até a solução. O impacto de uma diminuição da frequência do *clock*, ou do tempo de espera por um recurso compartilhado no processador, é maior nos tempos de execução pequenos do que nos tempos de execução maiores, onde esses atrasos são diluídos no tempo total.

- A estimação do *speedup* exige que condições sejam satisfeitas para que a sua distribuição se aproxime da distribuição normal. A frequente ocorrência de dados discrepantes observada nos experimentos realizados pode levar a uma estimativa enviesada do *speedup* definido como uma razão de duas médias. A média é sensível a valores extremos, e a razão entre duas médias nem sempre tem uma média.

- Os métodos robustos modernos de análise dos dados, como o *trimmed mean*, foram aplicados na metodologia proposta. Existem outros métodos robustos que devem ser investigados, pois podem melhorar a metodologia proposta. Segundo Erceg-Hurn e Mirosevich (2008), testes estatísticos clássicos baseado em hipóteses, como análise de variância (ANOVA), teste t de *Student*, regressão linear pelo método dos mínimos quadrados, são baseados em suposições que são raramente satisfeitas por dados reais. O uso desses testes quando as suposições não são satisfeitas pode levar a valores-p imprecisos, aumentando o risco de rejeitar erroneamente a hipótese nula (isto é, concluir que um efeito real existe quando ele não existe–erro tipo I) e o poder de detectar efeitos genuínos (potência do teste) é reduzido. A maioria dos textos que argumentam que os testes estatísticos clássicos são robustos quando o planejamento é balanceado, apenas consideram pequenos desvios de normalidade e homoscedasticidade[1], e não consideram os pontos discrepantes que são comuns nos dados reais. Os autores referem-se especificamente aos dados da área da psicologia, mas nos experimentos relatados no Capítulo 7 também foram encontrados pontos discrepantes.

- O planejamento fatorial 2^k supõe que os fatores e interações são aproximadamente

[1]Homoscedasticidade é o termo para designar variância constante dos erros.

lineares nos intervalos definidos pelos níveis dos fatores. Essa é uma limitação da metodologia proposta. Na avaliação da função Rastrigin, a verificação visual dos resíduos do modelo ajustado sugere uma possível não-linearidade, que deve ser investigada futuramente. Efeitos quadráticos requerem outro tipo de planejamento, como o planejamento composto central. Esse é um trabalho a ser futuramente realizado.

Bibliografia

B. Adenso-Díaz e M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

S. Akhter e J. Roberts. *Multi-Core Programming*. Intel Corporation, first edition edition, 2006.

E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

E. Alba. *Parallel metaheuristics: a new class of algorithms*, volume 47 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, 2005.

E. Alba e G. Luque. Evaluation of parallel metaheuristics. In *Workshop on Empirical Methods for the Analysis of Algorithms*, PPSN-EMAA'06, pages 9–14, Reykjavik, Iceland, September 2006.

E. Alba, G. Luque, e S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013. ISSN 1475-3995. doi: 10.1111/j.1475-3995.2012.00862.x.

ARM Ltd. The ARM cortex-a9 processors. Technical report, ARM Ltd. White Paper, September, 2009.

K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, e K. A. Yelick. The landscape of parallel computing reserarch: A view from berkeley. Technical Report UCB/EECS-2006-183, Electrical Engineering and Computer Sciences University of California at Berkeley, 2006.

K. A. Baggerly e K. R. Coombes. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *The Annals of Applied Statistics*, pages 1309–1334, 2009.

P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. L. Lusk, R. Thakur, e J. L. Träff. MPI on Millions of Cores. *Parallel Processing Letters (PPL)*, 21(1):45–60, Mar. 2011.

P. Balaprakash, M. Birattari, e T. Stutzle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics*, pages 108–122, 2007.

R. S. Barr e B. L. Hickman. Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. *ORSA Journal on Computing Winter*, 5:2–18, 1993.

R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, e W. R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, first edition, 2006. ISBN 3540320261.

T. Bartz-Beielstein. Spot: An r package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. *CoRR*, abs/1006.4645, 2010.

T. Bartz-Beielstein e M. Preuss. The future of experimental research. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, e M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 17–49. Springer-Verlag, Berlin, 2010.

T. Bartz-Beielstein, M. Chiarandini, L. Paquete, e M. Preuss. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer-Verlag, Berlin, 2010. ISBN 978-3-642-02537-2.

A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, e A. Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 29–44, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: http://doi.acm.org/10.1145/1629575.1629579.

E. Betti, D. P. Bovet, M. Cesati, e R. Gioiosa. Hard real-time performances in multiprocessor-embedded systems usingasmp-linux. *EURASIP J. Embedded Syst.*, 2008:10:1–10:16, April 2008. ISSN 1687-3955. doi: http://dx.doi.org/10.1155/2008/582648.

M. Birattari. *The problem of tuning metaheuristics as seen from a maching learning perspective*. PhD thesis, Universite Libre de Bruxelles, 2004.

M. Birattari e M. Dorigo. How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? *Optimization Letters*, 1(3):309–311, 2006.

G. Blake, R. G. Dreslinski, e T. Mudge. A survey of multicore processors. *Signal Processing Magazine, IEEE*, 26(6):26–37, Oct. 2009. doi: 10.1109/MSP.2009.934110.

G. E. P. Box e D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society*, 26(2):211–252, 1964.

G. E. P. Box, J. S. Hunter, e W. G. Hunter. *Statistics for experimenters: design, discovery, and innovation*. John Wiley & Sons, Hoboken, 2 edition, 2005.

S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, e Z. Zhang. Corey: an operating system for many cores. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 43–57, Berkeley, CA, USA, 2008. USENIX Association.

S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, e N. Zeldovich. An analysis of linux scalability to many cores. In *OSDI 2010: Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation*, 2010.

M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, e D. I. August. Revisiting the sequential programming model for the multicore era. *IEEE Micro*, 28(1):12–20, 2008. ISSN 0272-1732. doi: http://dx.doi.org/10.1109/MM.2008.13.

D. Buntinas, G. Mercier, e W. Gropp. Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem. *Parallel Computing*, 33(9):634 – 644, 2007. ISSN 0167-8191. doi: 10.1016/j.parco.2007.06.003. Selected Papers from EuroPVM/MPI 2006.

J. R. Busemeyer e Y.-M. Wang. Model comparisons and model selections based on generalization criterion methodology. *Journal of Mathematical Psychology*, 44(1): 171–189, 2000.

E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles*, 10, 1998.

E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 0792372212.

E. Carrano, E. Wanner, e R. Takahashi. A multicriteria statistical based comparison methodology for evaluating evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 15(6):848 –870, dec. 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2069567.

J. Charles, P. Jassi, N. S. Ananth, A. Sadat, e A. Fedorova. Evaluation of the intel® core i7 Turbo Boost feature. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 188–197. IEEE, 2009.

M. Chiarandini, L. Paquete, M. Preuss, e E. Ridge. Experiments on metaheuristics: methodological overview and open issues. Technical Report DMF-2007-03-003, The Danish Mathematical Society, 2007.

M. Coffin e M. J. Saltzman. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing*, 12:24–44, January 2000. ISSN 1526-5528. doi: 10.1287/ijoc.12.1.24.11899.

J. Cohen. *Statistical power analysis for the behavioral sciences : Jacob Cohen.* Lawrence Erlbaum, 2 edition, Jan. 1988. ISBN 0805802835.

S. P. Coy, B. L. Golden, G. C. Runger, e E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001. ISSN 1381-1231. 10.1023/A:1026569813391.

A. Czarn, C. MacNish, K. Vijayan, e B. TOPTurlach. Statistical exploratory analysis of genetic algorithms: the importance of interaction. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC 2004)*, volume 2, pages 2288 – 2295 Vol.2, june 2004a. doi: 10.1109/CEC.2004.1331182.

A. Czarn, C. MacNish, K. Vijayan, B. A. Turlach, e R. Gupta. Statistical exploratory analysis of genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8 (4):405 – 421, aug. 2004b. ISSN 1089-778X. doi: 10.1109/TEVC.2004.831262.

J. A. da Silva, O. A. C. Cortes, e J. C. da Silva. A java-based code generator for parallel evolutionary algorithms. In *1st BRICS Countries Congress on Computational Intelligence and 11th CBIC Brazilian Congress on Computational Intelligence*, 2013.

C. Daniel. Use of half-normal plots in interpreting factorial two-level experiments. *Technometrics*, 1(4):311–341, 1959.

J. Dongarra, T. Sterling, H. Simon, e E. Strohmaier. High-performance computing: Clusters, constellations, mpps, and future directions. *Computing in Science and Engg.*, 7(2):51–59, 2005. ISSN 1521-9615. doi: http://dx.doi.org/10.1109/MCSE.2005.34.

J. Dongarra, D. Gannon, G. Fox, e K. Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1), February 2007.

R. Duncan. A survey of parallel computer architectures. *Computer*, 23(2):5–16, 1990. doi: 10.1109/2.44900.

E. Díaz-Francés e F. Rubio. On the existence of a normal approximation to the distribution of the ratio of two independent normal random variables. *Statistical Papers*, 54(2):309–323, 2013. ISSN 0932-5026. doi: 10.1007/s00362-012-0429-2. URL `http://dx.doi.org/10.1007/s00362-012-0429-2`.

A. Eiben, R. Hinterding, e Z. Michalewicz. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124 –141, jul 1999. ISSN 1089-778X. doi: 10.1109/4235.771166.

A. E. Eiben e M. Jelasity. A critical note on Experimental Research Methodology in Experimental research methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, volume 5, pages 582–587, 2002. doi: 10.1109/CEC.2002.1006991.

A. E. Eiben e J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 2003. ISBN 3540401849. doi: 10.1007/978-3-662-05094-1.

A. E. Eiben e S. K. Smith. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1:19–31, 2011.

D. M. Erceg-Hurn e V. M. Mirosevich. Modern robust statistical methods: an easy way to maximize the accuracy and power of your research. *American Psychologist*, 63(7):591, 2008.

R. E. Ferreira. Implementação de algoritmos genéticos paralelos em uma arquitetura mpsoc. Master's thesis, UERJ, 2009.

A. Fog. *Random Number Generator Libraries*, 2010. URL `http://www.agner.org/random/`. 2008-2010, Instructions for the random number generator libraries on www.agner.org. GNU General Public License. Version 2.01. 2010-08-03. Accessed: 15 April 2011.

J. Fox. *Regression diagnostics: An introduction*, volume 79. Sage, 1991.

J. Fox e S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, second edition, 2011. URL `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion`.

H. C. Freitas, M. A. Z. Alves, N. B. Maillard, e P. O. A. Navaux. Ensino de arquiteturas de processadores multi-core através de um sistema de simulação completo e da experiência de um projeto de pesquisa. In *Workshop sobre Educação em Arquitetura de Computadores - WEAC/SBAC-PAD*, 2008.

D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1st edition, 1989.

J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, 1986.

J. L. Hennessy e D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Prulishers, San Francisco, 4th edition, 2006.

G. Hillar. Challenges in multi-core era - part 2. Techdoer Times, June 2009.

J. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42: 201–212, 1994.

J. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42, 1995. ISSN 1381-1231. 10.1007/BF02430364.

F. Hutter, H. H. Hoos, K. Leyton-Brown, e T. Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36: 267–306, September 2009. ISSN 1076-9757.

IBM Research Team. The Cell architecture. Acessado em 10 de janeiro de 2012.

E. H. Isaaks e R. M. Srivastava. *Applied geostatistics*. Oxford University Press, New York, 1989.

R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons Inc, New York, 1 edition, 1991.

D. Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, 5:215–250, 2002.

C. C. Kannas, C. A. Nicolaou, e C. S. Pattichis. A parallel implementation of a multi-objective evolutionary algorithm. In *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on*, pages 1–6. IEEE Computer Society, 2009.

H. Kim e R. Bond. Multicore software technologies. *IEEE Signal Processing Magazine*, 26(6):80–89, November 2009.

D. S. Knysh e V. M. Kureichik. Parallel genetic algorithms: a survey and problem state of the art. *Journal of Computer and Systems Sciences International*, 49(4): 579–589, 2010. doi: 10.1134/S1064230710040088.

M. H. Kutner, J. Neter, C. J. Nachtsheim, e W. Li. *Applied linear statistical models*. The McGraw-Hill/Irwin series operations and decision sciences. McGraw-Hill Irwin, New York, 5 edition, 2005.

E. G. M. Lacerda e A. C. P. L. F. Carvalho. Introdução aos algoritmos genéticos. *Revista de Informática Teórica e Aplicada*, 3:7–39, 2002.

J. Lässig e D. Sudholt. The benefit of migration in parallel evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1105–1112, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0072-8. doi: http://doi.acm.org/10.1145/1830483.1830687.

F. Leisch. Sweave, part i: Mixing r and latex. *R News*, 2(3):28–31, 2002.

G. Luque e E. Alba. *Parallel Genetic Algorithms Theory and Real World Applications*, volume 367 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 1st edition, 2011. doi: 10.1007/978-3-642-22084-5.

M. D. MacCool. Scalable programming models for massively multicore processors. *Proceedings of the IEEE*, 96(5):816–831, may 2008.

A. Marowka. Think parallel: Teaching parallel programming today. *IEEE Distributed Systems Online*, 9(8):1–8, 2008.

G. Marsaglia. The mother of all random generators. Published by Bob Wheeler in sci.stat.consult and sci.math.num-analysis in the name of George Marsaglia, 28th October 1994. Code and comments in: http://www.stat.berkeley.edu/classes/s243/mother.c, 1994.

A. Mazouz, D. Barthou, et al. Analysing the variability of openmp programs performances on multicore architectures. In *Fourth Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG-2011), Heraklion, January 23*, 2011.

C. C. McGeoch. Feature article - toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996.

J. M. P. Memória. *Breve história da estatística*. Embrapa Comunicação para Transferência de Tecnologia: Embrapa-Secretaria de Administração Estratégica, 2004.

R. E. Mercer e J. Sampson. Adaptive search using a reproductive meta-plan. *Kybernetes*, 7(3):215–228, 1978.

Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, New York, 3rd edition, 1996.

C. M. Micheel, S. J. Nass, G. S. Omenn, et al. *Evolution of translational omics: lessons learned and the path forward*. National Academies Press, 2012.

D. Montgomery e G. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Danvers, 3 edition, 2003. ISBN 0-471-20454-4.

D. Montgomery e G. Runger. *Estatística Aplicada e Probabilidade para Engenheiros*. LTC, 2009. ISBN 9788521616641.

D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley and Sons, Hoboken, 7th edition, 2009.

S. K. Moore. Multicore CPUs: Processor proliferation. *IEEE Spectrum*, January 2011.

J.-B. Mouret e S. Doncieux. Sferesv2: Evolvin' in the multi-core world. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8, 2010.

MPICH. MPICH: High-Performance Portable of the Message Passing Interface (MPI) standard (MPI-1, MPI-2 and MPI-3). Argonne National Laboratory, 2011. [online] `http://www.mcs.anl.gov/mpi/mpich2`. Accessed: 20 May 2011.

H. Mühlenbein. Darwin's continent cycle theory and its simulation by the prisoner's dilemma. *Complex Systems*, 5:459–478, 1991.

A. Munawar, M. Wahib, M. Munetomo, e K. Akama. A survey: Genetic algorithms and the fast evolving world of parallel computing. In *High Performance Computing and Communications, 10th IEEE International Conference on*, volume 0, pages 897–902, Los Alamitos, CA, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3352-0. doi: http://doi.ieeecomputersociety.org/10.1109/HPCC.2008.77.

V. Nannen e A. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the International Joint Conference on Atrificial Intelligence (IJCAI'07)*, pages 975–980. International Joint Conferences on Artificial Intelligence, AAAI Press, 2007.

J. Neter, M. H. Kutner, C. J. Nachtsheim, e W. Wasseman. *Applied Linear Statistical Models*. McGraw-Hill, 4 edition, 1996.

B. B. Neto, I. S. Scarminio, e R. E. Bruns. *Como Fazer Experimentos - Pesquisa e Desenvolvimento na Ciência e na Indústria*. Editora UNICAMP, 4th edition, 2010.

NVIDIA Corporation. Nvidia's next generation cuda compute architecture fermi. Technical report, NVIDIA Corporation, 2009.

K. Olukotun e L. Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005. ISSN 1542-7730. doi: http://doi.acm.org/10.1145/1095408.1095418.

M. S. Pais, I. S. Peretta, G. F. M. Lima, J. A. Tavares, H. X. Rocha, e K. Yamanaka. Análise de fatores determinantes no desempenho dos algoritmos genéticos paralelos em processadores multinúcleos. In *X Congresso Brasileiro de Inteligência Computacional*, 2011.

M. S. Pais, I. S. Peretta, K. Yamanaka, e E. R. Pinto. Factorial design analysis applied to the performance of parallel evolutionary algorithms. *Journal of the Brazilian Computer Society*, 20(6), 2014. ISSN 1678-4804. doi: 10.1186/1678-4804-20-6.

D. Patterson. The trouble with multicore. *IEEE Spectrum*, 47:28 – 32, 2010. ISSN 0018-9235. doi: 10.1109/MSPEC.2010.5491011.

A. Petrovski, A. E. I. Brownlee, e J. A. W. McCall. Statistical optimisation and tuning of GA factors. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 1, pages 758–764, 2005. doi: 10.1109/CEC.2005.1554759.

A. F. Pinho, J. A. B. Montevechi, e F. A. S. Marins. Análise da aplicação de projeto de experimentos nos parâmetros dos algoritmos genéticos. *Sistemas & Gestão*, 2: 319–331, 2007. ISSN 19805160.

J. H. M. Pinho, M. F. de Almeida P. Rocha, e J. L. F. Sobral. Pluggable parallelization of evolutionary algorithms applied to the optimization of biological processes. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 0: 395–402, 2010. ISSN 1066-6192. doi: http://doi.ieeecomputersociety.org/10.1109/PDP.2010.89.

C. G. Qiao, G. R. Wood, C. D. Lai, e D. W. Luo. Comparison of two common estimators of the ratio of the means of independent normal variables in agricultural research. *Journal of Applied Mathematics and Decision Sciences*, 2006, 2006. doi: 10.1155/JAMDS/2006/78375.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `http://www.R-project.org/`.

C. R. Rao. Statistics: a technology for the millennium. *International Journal of Mathematical and Statistical Sciences*, 8:5–25, 1999.

R. L. Rardin e R. Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7:261–304, May 2001. ISSN 1381-1231. doi: 10.1023/A:1011319115230.

T. Rauber e G. Rünger. *Parallel Programming for Multicore and Cluster Systems*. Springer-Verlag Berlin Heidelberg, 1st edition, 2010. ISBN 9783642048661.

S. Ribas, M. H. P. Perché, I. M. Coelho, P. L. A. Munhoz, M. J. F. Souza, e A. L. L. Aquino. Mapi: Um framework para paralelização de algoritmos. *Learning and Nonlinear Models (LNLM) Journal of the BrazilianNeural Network Society*, 8:163–173, 2010.

E. Ridge e D. Kudenko. Screening the parameters affecting heuristic performance. In *In Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2007a.

E. Ridge e D. Kudenko. Tuning the performance of the mmas heuristic. In *Proceedings of the 2007 international conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics*, SLS'07, pages 46–60, Berlin, Heidelberg, 2007b. Springer-Verlag. ISBN 978-3-540-74445-0.

R. Roy. *A primer on the Taguchi method*. SME-Society of Manufacturing Engineers, 2nd edition, 2010.

M. Ruciński, D. Izzo, e F. Biscani. On the impact of the migration topology on the island model. *Parallel Computing*, 36:555–571, October 2010. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/j.parco.2010.04.002.

M. Saito e M. Matsumoto. Simd-oriented fast mersenne twister: a 128-bit pseudo-random number generator. In A. Keller, S. Heinrich, e H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 607–622. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-74496-2.

M. Sato, Y. Sato, e M. Namiki. Proposal of a multi-core processor architecture for effective evolutionary computation. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1321–1322, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0072-8. doi: http://doi.acm.org/10.1145/1830483.1830723.

C. Schepke e N. Maillard. Programming languages and models for parallel multi-level architecture. In *IV Workshop de Processamento Paralelo e Distribuído*, Porto Alegre, Brazil, 2008.

M. Shahsavar, A. A. Najafi, e S. T. A. Niaki. Statistical design of genetic algorithms for combinatorial optimization problems. *Mathematical Problems in Engineering*, 2011, 2011. doi: 10.1155/2011/872415.

X. Shengjun, G. Shaoyong, e B. Dongling. The analysis and research of parallel genetic algorithm. In *4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pages 1–4. IEEE Computer Society, 2008.

S. Smit e A. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 399 –406, may 2009. doi: 10.1109/CEC.2009.4982974.

J. Spall. Factorial design for efficient experimentation. *Control Systems, IEEE*, 30(5): 38 –53, oct. 2010. ISSN 1066-033X. doi: 10.1109/MCS.2010.937677.

G. Speyer, N. Freed, R. Akis, D. Stanzione, e E. Mack. Paradigms for parallel computation. In *DoD HPCMP Users Group Conference*. IEEE Computer Society, 2008.

D. M. Steinberg e W. G. Hunter. Experimental design: review and comment. *Technometrics*, 26(2):71–97, 1984.

K. Táng, X. Lǐ, P. N. Suganthan, Z. Yáng, e T. Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Héféi, Ānhuī, China, Jan. 8, 2010.

M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 2005. ISBN 3540241930.

S.-A.-A. Touati, J. Worms, e S. Briais. The speedup-test: a statistical methodology for programme speedup analysis and computation. *Concurrency and Computation: Practice and Experience*, 25(10):1410–1426, 2013. ISSN 1532-0634. doi: 10.1002/cpe.2939. URL `http://dx.doi.org/10.1002/cpe.2939`.

W. N. Venables e B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

D. Wentzlaff e A. Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.*, 43:76–85, April 2009. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1531793.1531805.

D. Whitley, S. Rana, e R. Heckendorn. Island model genetic algorithms and linearly separable problems. In D. Corne e J. Shapiro, editors, *Evolutionary Computing*, volume 1305 of *Lecture Notes in Computer Science*, pages 109–125. Springer Berlin / Heidelberg, 1997. ISBN 978-3-540-63476-8. 10.1007/BFb0027170.

R. R. Wilcox e H. Keselman. Modern robust data analysis methods: measures of central tendency. *Psychological methods*, 8(3):254, 2003.

D. Wolpert e W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

D. H. Wolpert e W. G. Macready. No free lunch theorems for search. Technical Report SFI–TR–95–2–010, Santa Fe Institute, 1996.

C.-F. J. Wu e M. Hamada. *Experiments: planning, analysis, and parameter design optimization*. John Wiley & Sons, 1st edition edition, 2000.

Y. Xie. *Dynamic Documents with R and knitr*. CRC Press, 2013a.

Y. Xie. knitr: A general-purpose package for dynamic report generation in R. *R package version*, 1, 2013b.

A. Zeileis e T. Hothorn. Diagnostic checking in regression relationships. *R News*, 2(3): 7–10, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

F. J. V. Zuben. Computação evolutiva: Uma abordagem pragmática. In *Anais da I Jornada de Estudos em Computação de Piracicaba e Região (1a JECOMP)*, pages 25–45, 2000.

Apêndice A

Outros Trabalhos Relacionados a AGPs

Trabalhos publicados recentemente e que estão relacionados a aplicações, propostas de novos algoritmos, e análises teóricas dos AGPs são descritas a seguir.

1. Sato et al. (2010) mostram que a recente expansão do uso de processadores multicores nos computadores de uso geral trouxe consigo um paralelismo numa escala muito menor e de custo mais baixo do que o dos computadores massivamente paralelos. Nesse cenário, surgiram pesquisas de métodos de conversão da computação evolucionária para o paralelismo nos processadores multicores de propósito geral, e considera que é também importante que as arquiteturas do processadores multicores levem em consideração as características da computação evolutiva e propõe indicadores para a seleção entre os processadores existentes os que são compatíveis com essas características. Apresentam uma arquitetura multicore onde o acesso a memória local de qualquer core pode ser realizado diretamente por outro core sem passar pela memória principal. Esta arquitetura favorece a computação evolutiva. Faz a comparação da arquitetura proposta com um processador de núcleo único e com processadores multicores atuais. Concluem que o desempenho da arquitetura proposta possui melhor desempenho e há redução no consumo de energia.

2. Em (Knysh e Kureichik, 2010), a partir da premissa de que um AG paralelo no modelo de ilhas com número grande de conexões entre ilhas encontra soluções de mesma qualidade que o AG sequencial, os autores sugerem que há questões ainda não resolvidas: quantas conexões são necessárias para que o AG paralelo se comporte de forma diferente do AG sequencial; qual o custo da conexão; se o custo da

conexão é alto, vale a pena paralelizar o AG. Propõem um modelo de algoritmo genético distribuído que responda a essas questões e voltado para plataformas com multiprocessadores. Os testes foram realizados em regime assíncrono, em um computador com processador AMD dual-core, 1 GB RAM, Windows XP. Afirmam que os resultados demonstram a possibilidade do uso eficiente de algoritmos genéticos distribuídos em processadores multicores.

3. Pinho et al. (2010) apresentam o desenvolvimento de versão paralela de um *framework* sem modificá-lo, apenas adicionando módulos que podem ser configurados e que modificam o comportamento do código original para o padrão paralelo.

4. Em (Mouret e Doncieux, 2010) é mostrado o *Sferesv2*, um *framework* em C++ projetado para executar com rapidez programas da computação evolucionária em computadores multicore. Os seus princípios norteadores são: projeto baseado em otimizações direcionadas para multicores; oferecer implementações atualizadas e selecionadas de algoritmos evolucionários (EA) e EAs multiobjetivos; ser abstrato e eficiente. Os testes mostram que a execução em multicores é mais rápida mas depende do tempo de execução da função de aptidão utilizada.

5. Em (Kannas et al., 2009) é feita a comparação do programa sequencial e do paralelo rodando em processador multicore: Core 2 Duo E8400 3.0 GHz 4 Gbytes RAM, implementado em Phython, uma linguagem interpretada que possui vários módulos de terceiros que podem ser adicionados. Phython permite que se implemente o paralelismo através de *threads* e através de processos. Primeiro foi implementado o modelo mestre-escravo usando *threads*. Foram encontrados problemas com o desempenho pois não é permitido mais de um *thread* por vez ativo no Interpretador Python e a versão com *threads* foi abandonada. A segunda implementação usou processos e o modelo com subpopulações: as subpopulações evoluem independentes por um número de iterações definido pelo usuário (normalmente 10% do número total de iterações que o algoritmo irá rodar). Quando todas as subpopulações completam a sua evolução, seus resultados são reunidos gerando uma nova população. Os testes apresentaram um aumento de velocidade de 1.6 em um dual-core. Futuramente será testado em um quadicore.

6. Em (Ferreira, 2009) é apresentada a implementação de um AG paralelo utilizando

o modelo das ilhas, para sistemas embutidos multiprocessados - MPSoC (Multi-Processor System-on-Chip). Usou o HMPS (Hermes MultiProcessor System-on-Chip) de domínio público. No AG paralelo do modelo de ilhas, cada processador do sistema embutido é responsável pela evolução de uma população independente das demais. Diferentes topologias lógicas, tais como anel, vizinhança e broadcast, são analisadas na fase de migração de indivíduos.

7. Ribas et al. (2010) implementa a abstração MapReduce na linguagem C++. MaPI auxilia a implementação de uma aplicação paralela sem se preocupar com a forma de comunicação entre os processos ou como o sistema fará a paralelização. A implementação do usuário pode ser sequencial. Mostra a paralelização de um algoritmo heurístico de otimização aplicado ao problema do caixeiro viajante. Os resultados mostram ganho de performance. A qualidade da solução é objetivo secundário. PC Pentium Core 2 Quad (Q6600), com 2,4 GHz, 8 GB of RAM, Linux Ubuntu 8.10 Kernel 2.6.24-25.

8. Em (Shengjun et al., 2008) são apresentados quatro tipos de modelos de AGPs, suas características e os seus parâmetros. Os autores comentam alguns critérios de avaliação de AGPs, como o *speedup* e a aplicação da lei de *Amdahls*.

9. Em (Munawar et al., 2008), os autores trazem um levantamento breve sobre AGs paralelos nas modernas arquiteturas paralelas: *grids*, *clusters* e multicores. Listam os desafios que os desenvolvedores de AGs tem que resolver perante os paradigmas paralelos modernos: ser tolerante a falhas, suporte a *late-binding*, saber aproveitar as vantagens das arquiteturas multicore, suportar a interconexão de aplicações distribuídas *loosy coupled*, tolerar atrasos de comunicação, poder contar com acesso a fonte de dados externos quando for necessário. Ressaltam a necessidade da construção de um ambiente unificado para o desenvolvimento de algoritmos paralelos. Propõem o uso de máquinas virtuais para superar os problemas com a heterogeneidade dos sistemas atuais e apresentam um algoritmo genético paralelo hierárquico que se adéqua aos *grids*.

10. da Silva et al. (2013) apresentam uma ferramenta que gera código automático para AG e AE paralelos e síncronos usando os modelos mestre-escravo, ilhas e celular.

Apêndice B

Mais Sobre Processadores Multicore

B.1 Projetos de Processadores Multicore

Segundo Rauber e Rünger (2010), existem vários projetos diferentes de processadores *multicore*, com diferentes números de núcleos, estrutura e tamanho da(s) *cache(s)*, forma de acesso do núcleo à(s) *cache(s)*, e uso de componentes heterogêneos. De modo geral podem ser classificados em projetos hierárquicos, *pipelined* e baseado em rede.

B.1.1 Projeto Hierárquico

Os processadores *multicore* com projeto hierárquico compartilham múltiplas *caches*. As *caches* são organizadas em configuração do tipo árvore e o tamanho da *cache* aumenta à medida que se aproxima da raiz.

O projeto hierárquico é o mais frequente entre os processadores *multicore* comerciais, como o IBM Power6, as famílias de processadores Intel Xeon e AMD Opteron e processadores Sun Niagara, e também em unidades de processamento gráfico (GPUs), como a Nvidia GeForce 8800, com 128 *stream processors* (SP) a 1.35 GHz organizado em 8 *texture/processor cluster* (TPC) tal que cada TPC contém 16 SPs.

B.1.2 Projeto Pipelined

No projeto *Pipelined*, os elementos de dados são processados por múltiplos núcleos em modo *pipelined*. Os dados entram no processador via uma porta de entrada e são passados sucessivamente através dos vários núcleos até que os dados deixem o último núcleo e o *chip* através de uma porta de saída. Cada núcleo executa passos específicos do processamento em cada dado.

O projeto *Pipelined* favorece aplicações em que os mesmos comandos devem ser aplicados a uma longa sequência de dados. Processadores de rede usados em roteadores e processadores gráficos trabalham dessa forma. Exemplo de processador com projeto *Pipelined* é o processador Xelerator X10 e X11 utilizados no processamento de pacotes de rede de forma *pipelined* dentro do chip. O Xelerator X11 contém até 800 núcleos separados que são arranjados em fila lógica linear. Os pacotes de rede a serem processados entram no chip através de várias portas de entrada em um lado do chip, são sucessivamente processados pelos núcleos e então saem do chip.

B.1.3 Projeto Baseado em Rede

No projeto *baseado em rede*, os núcleos do processador e suas *caches* locais e memórias são conectados por uma rede de interconexão com outros núcleos do *chip*. A transferência de dados entre núcleos é executada através da rede de interconexão. Essa rede pode também possuir suporte para a sincronização dos núcleos. Interfaces fora do chip ocorrem por núcleos especializados ou portas de DMA (*direct memory access*). Um exemplo de projeto baseado em rede é o processador Intel Teraflop. O protótipo do processador Intel Teraflop possui 80 núcleos que estão organizados em malha (*mesh*) 8x10. Cada núcleo executa operações em ponto-flutuante, possui *cache* local, e pode transferir dados entre núcleos e a memória principal. Existem núcleos adicionais para o processamento de imagens, codificação e processamento gráfico.

B.2 Outras Classificações das Arquiteturas Multicores

Existem outras maneiras de se classificar as arquiteturas *multicore*. No trabalho de Blake et al. (2009) são apresentadas classificações baseadas em cinco atributos mais significantes dos *multicores*: a classe de aplicação, a relação consumo de energia/desempenho, os elementos de processamento e o sistema de memória. O termo elemento de processamento nada mais é do que o núcleo de processamento. Como esse é o termo que foi utilizado em (Blake et al., 2009), no texto a seguir a mesma terminologia é utilizada.

B.2.1 Classe de Aplicação

Quando uma arquitetura é projetada para atender a um domínio específico de aplicação, ela pode ser construída para responder melhor às características desse domínio.

A arquitetura será eficiente para esse domínio, porém ela pode ser ineficiente para outros. Os domínios de aplicações podem ser dominados pelo processamento de dados ou pelo processamento de controles.

Entre as aplicações dominadas pelo processamento de dados, tem-se a rasterização de gráficos, o processamento de imagens e de áudio, o processamento de comunicações *wireless* e os algoritmos para processamento de sinais. A computação baseia-se tipicamente em uma sequência de operações sobre um fluxo (*stream*) de dados onde pouca ou nenhuma informação é reutilizada e uma alta taxa de transferência (*throughput*) é necessária. Em geral, essas operações podem ser feitas em paralelo e por isso favorecem uma arquitetura com grande número de elementos de processamento, mantendo-se a relação entre o consumo de energia e o desempenho.

As aplicações onde o processamento de controles é dominante incluem a compressão/descompressão de arquivos, os processamentos em rede e a execução de operações tradicionais. Os programas tendem a possuir muitos desvios condicionais que dificultam a sua paralelização. Nesse caso, essas aplicações tem melhor desempenho com um número modesto de elementos de processamento de propósito geral.

Na maioria dos casos, nenhuma aplicação se ajusta exatamente a uma dessas duas divisões, mas essa divisão é importante para entender como diferentes arquiteturas *multicore* podem afetar o desempenho de aplicações com características específicas.

B.2.2 Relação Consumo de Energia / Desempenho

Muitos dispositivos possuem requisitos de consumo de energia e desempenho restritivos. Um telefone celular com suporte a execução de vídeos, deve consumir pouca energia e manter certas características de qualidade. Telefones celulares e computadores portáteis possuem baterias com tamanho e vida útil reduzidos; *data centers* que dão suporte à computação em nuvens também tem o consumo de energia como atributo importante. Para esses casos, o modelo de processador *multicore* de propósito geral é o mais indicado.

B.2.3 Elementos de Processamento

Os elementos de processamento podem ser abordados de duas maneiras: pela sua arquitetura e pela sua microarquitetura. A arquitetura está relacionada ao seu conjunto de instruções (em inglês, ISA - *instruction set architecture*) e define a interface entre o

hardware e o software. Já a microarquitetura corresponde à implementação do conjunto de instruções.

O ISA de cada núcleo de processadores *multicore* convencionais é tipicamente um ISA legado dos processadores mais antigos com pequenas modificações para suportar paralelismos, como a inclusão de instruções atômicas para sincronização. A vantagem desses ISAs que mantém o legado das instruções é a sua compatibilidade tanto com as implementações quanto com as ferramentas de programação já existentes. Os fabricantes tem continuamente melhorado o desempenho de operações mais comuns através de extensões adicionadas ao conjunto. A Intel adicionou os subconjuntos MMX, MMX2 e SS1-4 para melhorar o desempenho dos processamentos multimídia. Outros fabricantes seguiram o mesmo caminho. Essas instruções empregam partes especializadas do hardware em sua execução e conseguem melhorar a relação entre consumo de energia e desempenho.

A microarquitetura de cada elemento de processamento é responsável em muitos aspectos pelo desempenho e pelo consumo de energia de um *multicore* e é normalmente associada ao domínio de aplicação para o qual o processador *multicore* é projetado. Muitas vezes é vantajoso combinar diferentes tipos de elementos de processamento em uma mesma arquitetura constituindo uma organização heterogênea onde, por exemplo, um processador de controle pode ser usado para comandar as atividades de um grupo de núcleos mais simples dedicados ao processamento de dados. Um dos benefícios dessa organização é reduzir o consumo de energia sem perder desempenho. Entretanto, o modelo de programação dessa arquitetura é normalmente mais complicado.

O tipo mais simples de elemento de processamento faz o despacho das instruções *em ordem*, ou seja, na ordem em que elas estão no programa. Esse modelo permite duas abordagens para se obter melhor desempenho: múltiplos *pipelines* podem ser adicionados para ler e despachar mais de uma instrução em paralelo, criando um elemento de processamento *superscalar*; e aumento do número de estágios do *pipeline*, reduzindo a lógica contida em cada estágio. Os elementos de processamento com despacho *em ordem* são fisicamente menores, consomem menos energia e podem ser facilmente combinados em grandes quantidades para atender aplicações que possuem alto paralelismo de *thread* (em inglês, TLP - *thread level parallelism*) e poucos trechos de execução sequencial. O fato do processamento gráfico ser altamente paralelo com poucos trechos sequenciais permite que os *multicores* da série GeForce da NVIDIA com arquitetura

Fermi (NVIDIA Corporation, 2009) reúnam 512 núcleos *em ordem*.

A execução *fora de ordem* é empregada para explorar o paralelismo ao máximo na execução de códigos sequenciais em núcleos superescalares. O objetivo é encontrar dinamicamente uma sequência em que as operações possam ser executadas de modo a manter o *pipeline* com o máximo de ocupação. O mecanismo de agendamento dinâmico da execução exige circuitos muito complexos e que consomem muita energia. Por serem maiores e consumirem mais energia, na prática, poucos núcleos *fora de ordem* podem ser combinados. Núcleos *fora de ordem* são apropriados para aplicações que possuem comportamentos de diversos tipos e requerem alto desempenho, ou seja, aplicações dominadas por estruturas de controle com grandes trechos de código sequencial e que possuem um TLP moderado. Um exemplo é o processador ARM Cortex A9 (ARM Ltd., September, 2009), que possui núcleos *fora de ordem* e é usado em *netbooks*.

Uma alternativa para evitar a complexidade da execução *fora de ordem* e melhorar o desempenho de arquiteturas superescalares é usar o modelo de única instrução sobre múltiplos dados (SIMD) ou o de instruções com palavras muito longas (em inglês, VLIW - *very long instruction word*). A arquitetura SIMD utiliza registradores maiores que são divididos em partes de informação menores para serem processadas simultaneamente pela mesma instrução. Um exemplo de aplicação que se beneficia com essa arquitetura é o cálculo do produto escalar entre dois vetores, onde cada par de elementos é processado em paralelo, ou seja, aplicações onde o processamento de dados é intensivo e independente. O processador IBM Cell (IBM Research Team) emprega múltiplos núcleos SIMD customizado para a execução de aplicações dominadas pelo processamento de dados.

As instruções VLIW podem ser usadas para amenizar as limitações da arquitetura SIMD onde apenas uma instrução pode operar sobre os dados. Assim, múltiplos *pipelines* são usados mas a complexidade do controle é deixada a cargo do compilador ou do programador que devem fazer o agrupamento das instruções em pacotes que possam ser executados em paralelo sem ferir as restrições de dados ou de controle. O desempenho dessa arquitetura pode ser afetado se o compilador (ou programador) não conseguir encontrar um nível satisfatório de paralelismo. As arquiteturas VLIW e SIMD possuem boa relação entre desempenho e consumo de energia, porém são adequadas somente para aplicações com grande número de operações independentes que podem ser identificadas e exploradas pelo compilador ou programador.

B.3 MPI - Conceitos básicos

O MPI especifica algumas funções básicas:

- funções de gerência de processos: iniciar (MPI_Init), finalizar (MPI_Finalize), determinar o número de processos (MPI_Comm_size), identificar processos (MPI_Comm_rank);
- funções de comunicação ponto-a-ponto (*Point-to-Point*): enviar (MPI_Send) e receber (MPI_Recv) mensagens entre dois processos;
- funções de comunicação de grupos: *broadcast* (MPI_Bcast, MPI_Gather, MPI_Scatter, MPI_Reduce), sincronizar processos (MPI_Barrier).

Alguns termos comumente utilizados em MPI são descritos a seguir:

- *Buffering*: cópia temporária de mensagens entre endereços de memória efetuada pelo sistema como parte de seu protocolo de transmissão. A cópia ocorre entre o *buffer* do usuário (definido pelo processo) e o *buffer* do sistema (definido pela biblioteca);
- *Blocking*: uma rotina de comunicação é *blocking*, quando a finalização da execução da rotina é dependente de certos eventos (espera por determinada ação, antes de liberar a continuação do processamento);
- *Non-blocking*: uma rotina de comunicação é *non-blocking*, quando a finalização da execução da rotina, não depende de certos eventos.
- Síncrono: comunicação na qual o processo que envia a mensagem, não retorna a execução normal enquanto não haja um sinal do recebimento da mensagem pelo destinatário;
- Assíncrono: comunicação na qual o processo que envia a mensagem, não espera que haja um sinal de recebimento da mensagem pelo destinatário.
- *Communicator*: define uma coleção de processos (*group*), que poderão se comunicar entre si. As funções do MPI exigem que seja especificado um *communicator* como argumento. MPI_COMM_WORLD é o *comunicator* pré-definido que inclui todos os processos definidos pelo usuário numa aplicação MPI.

- *Rank*: identificação única que o processo recebe ao ser iniciado. Essa identificação é contínua e inicia no zero até $N - 1$ processos, onde N é o total de processos.

- *Group*: é um conjunto ordenado de N processos. Todo e qualquer grupo é associado a um *communicator* e inicialmente todos os processos são membros de um grupo com um *communicator* já pré-estabelecido (MPI_COMM_WORLD).

- *Application Buffer*: é um endereço de memória onde se armazena dados que o processo necessita enviar ou receber.

- *System Buffer*: é um endereço de memória reservado pelo sistema para armazenar mensagens. Dependendo do tipo de operação de *send/receive*, os dados no *application buffer* devem ser copiados de/para o *system buffer* (*Send Buffer* e *Receive Buffer*). Neste caso teremos uma comunicação assíncrona.

- *Blocking Communication*: uma rotina de comunicação é dita *blocking* se a finalização da chamada depender de certos eventos. Por exemplo, numa rotina de envio o dado tem que ter sido enviado com sucesso, ou ter sido salvo no *system buffer* indicando que o endereço do *application buffer* pode ser reutilizado. Numa rotina de recebimento, o dado tem que ser armazenado no *system buffer*, indicando que esse dado pode ser utilizado.

- *Non-Blocking Communication*: uma rotina de comunicação é dita *non-blocking*, se a chamada retorna sem esperar qualquer evento que indique o fim ou o sucesso da rotina. Por exemplo, não espera pela cópia de mensagens do *application buffer* para o *system buffer*, ou a indicação do recebimento de uma mensagem.

  É responsabilidade do programador ter a certeza de que o *application buffer* esteja disponível para ser reutilizado. Este tipo de comunicação é utilizada para melhorar a relação entre computação e comunicação para efeitos de ganho de desempenho.

- *Standard Send*: operação básica de envio de mensagens usada para transmitir dados de um processo para outro.

- *Synchronous Send*: bloqueia até que ocorra um comando *receive* correspondente no processo de destino.

- *Buffered Send*: o programador cria um *buffer* para o dado antes dele ser enviado. Necessidade de se garantir um espaço disponível para um *buffer*, na incerteza do espaço do *System Buffer*.

- *Standard Receive*: operação básica de recebimento de mensagens usado para aceitar os dados enviados por qualquer outro processo. Pode ser *blocking* e *non-blocking*.

- *Return Code*: valor inteiro retornado pelo sistema para indicar a finalização da sub-rotina.

Comunicação *Point-to-Point*

Os componentes básicos de qualquer implementação MPI são as rotinas de comunicação *Point-to-Point*:

- Bloking:
  - Send: finaliza, quando o "buffer"de envio está pronto para ser reutilizado;
  - Receive: finaliza, quando o "buffer"de recebimento está pronto para ser reutilizado.

- Non-Bloking:
  - Send e Receive: retorna imediatamente, após envio ou recebimento de uma mensagem.

Comunicação Coletiva

As rotinas de comunicação coletivas são voltadas para coordenar grupos de processos. Existem basicamente três tipos de rotinas de comunicação coletiva:

- Sincronização:
  - Envio de dados: Broadcast, Scatter/Gather, All to All. Computação Coletiva: Min, Max, Add, Multiply, etc

B.4 Implicações da Tecnologia *Multicore*

As implicações da adoção da tecnologia *multicore* como estratégia de evolução dos processadores trouxe implicações em diversas áreas, como a programação e desenvolvimento de sistemas, nos sistemas operacionais e no currículo dos cursos de formação de profissionais que atuam nessas áreas. Nas subseções seguintes é apresentado um levantamento de trabalhos com temas relacionados às implicações mencionadas.

Na subseção B.4.1 estão os trabalhos que abordam o impacto da tecnologia *multicore* na programação e no desenvolvimento de sistemas. Os sistemas operacionais também sofreram alterações para se adequar e novos sistemas operacionais estão sendo projetados especificamente para os multicores, como mostram os trabalhos citados na subseção B.4.2. Todas as mudanças também são refletidas nos cursos das áreas de engenharia e informática, onde a formação de novos profissionais deve incluir conhecimentos relacionados a esses novos paradigmas, como mostram os trabalhos relacionados descritos na subseção B.4.3.

B.4.1 Programação e Desenvolvimento de Sistemas

A preocupação com as implicações que a adoção da tecnologia de processadores *multicore* como estratégia de evolução dos fabricantes de processadores é abordada por vários trabalhos recentes como (Moore, 2011; Patterson, 2010), e nos trabalhos abaixo:

1. Em (Olukotun e Hammond, 2005), os autores mostram que o mercado consumidor de processadores possui requisitos de performance diferentes: desempenho baseado na vazão de dados (*throughput*) e desempenho do tempo de latência (*latency*). O primeiro caso se aplica aos servidores de internet, onde o usuário não se importa de esperar um pouco mais, mas reclamará se o serviço requisitado não for executado. Com o aumento do número de usuários mais unidades de processamento foram adicionados aos servidores, ocupando maior espaço e consumindo cada vez mais energia. Os construtores de servidores começaram a economizar espaço adotando empacotando com maior densidade multiprocessadores que compartilham recursos físicos espaçosos, como discos rígidos e fontes. Houve também redução do consumo de energia com o compartilhamento de recursos que consumiam muita energia. Porém estas soluções estão se esgotando pois a maior densidade dos multiprocessadores acarreta na necessidade de sistemas de refri-

geração eficientes. Atualmente, busca-se resolver o problema de espaço físico e de consumo de energia com os *multicore* ou CMP - Chip Multiprocessor. Uma das primeiras experiencias com *multicores* foi o processador Niagara da Sun, com oito cores. A maioria das aplicações desktop se enquadram no segunda caso: o tempo de resposta de uma tarefa específica é mais importante do que o número de tarefas que podem ser executadas ao mesmo tempo. Para estes o ganho de performance com o processador *multicore* vai exigir o esforço dos programadores de programar aplicação de forma paralela. A maioria dos programadores não possui formação nem experiência com programação paralela.

2. Em (Asanovic et al., 2006), um grupo multidisciplinar de pesquisadores da Universidade de Berkeley reuniu-se por dois anos discutindo a mudança da indústria para os processadores *multicores*, aqueles que preservam os paradigmas de programação existentes através da compatibilidade binária, de mecanismo de *cache coherence* e a tendência de se dobrar o número de cores a cada nova geração. A conclusão do grupo é que essa abordagem de evolução para hardware e software paralelos pode funcionar para sistemas com 2 ou 8 cores, porém parece ter retornos insignificantes para sistemas com 16 ou 32 cores, uma vez que os resultados do paralelismo no nível de instruções também serão insignificantes. A experiência obtida com uso do paralelismo nos sistemas embarcados e nos sistemas de alto desempenho deve ser aproveitada para esse novo cenário.

3. O trabalho (Kim e Bond, 2009) traz a comparação entre várias linguagens quanto às suas capacidades (data parallel, task parallel, etc.) e seu desempenho em relação à performance e o esforço relativo a dificuldade de programação na área de processamento digital de sinais e de imagens em arquiteturas *multicore*. Usa o termo genérico de categorias de software para descrever as linguagens de programação paralela (*Unified Parallel C* - UPC, Sequoia, Co-Array Fortran, Titanium, Cilk, StreamIt, Chapel, Fortress, X10), a programação de propósito geral em GPUs (OpenGL, CUDA, Brook+, OpenCL), linguagens de alto nível (pMatlab, Star-P) e bibliotecas (PVL, VSIPL++, PVTOL).

4. Em (Bridges et al., 2008), os autores argumentam que como a maioria das aplicações não é paralela, a adição de mais cores ao processador não trará melhoria de desempenho. A experiência de décadas com a programação paralela explícita

tem se mostrado ser uma tarefa difícil. A paralelização manual é frequentemente otimizada para um número específico de cores ou uma plataforma de hardware específica. Essa estratégia consegue obter melhores performances mas é dificilmente portável. Surgiram novas linguagens (*Cilk, StreamIt, Atomos*) com propostas de tornar a programação paralela mais fácil porém dada a complexidade da programação paralela, os programadores frequentemente extraem somente o paralelismo fácil de ser identificado, deixando grande regiões com código sequencial, limitando o desempenho da aplicação. Pesquisas que tentaram extrair automaticamente *threads* a partir de programas *single-thread* sem a interferência do programador não sucederam, com exceção de alguns casos na área de aplicações científica. O trabalho propõe uma infraestrutura formada por análises e otimizações de compilação, inclusive para otimizações necessárias para o hardware. Essa infraestrutura é aplicada a todo *loop* do programa. Para evitar as restrições que outras técnicas de extração automática de *threads* encontraram são adicionadas anotações ao código sequencial que proveem informações ao compilador. O *benchmark* SPEC CINT2000 foi utilizado para testar a infraestrutura proposta. Mudando apenas 60 das 500000 linhas de código, foi obtida uma melhora de 454% no desempenho em um sistema com 32 cores.

5. Em (Dongarra et al., 2007), os autores relatam que: 1) o número de transistores em um processador continuará a dobrar em aproximadamente 18 meses, mas a velocidade do *clock* não continuará a crescer; 2) o número de pinos e largura de banda *bandwidth* do processador está chegando ao limite e 3) existirá uma forte tendência de tornar os sistemas de alto desempenho em sistemas híbridos. Cada um desses pontos traz implicações no desenvolvimento de programas, que são listadas a seguir:

   (a) Apesar das similaridades, processadores *multicores* não são equivalentes a multiprocessadores simétricos (SMP). Múltiplos cores em um mesmo processador podem ter vários níveis de *cache* e compartilham a mesma estrutura de interconexão. Para extrair o melhor desempenho dessa configuração de recursos significa que os programadores devem utilizar mais paralelismo a nível de *thread* e mecanismos eficientes de comunicação e sincronização entre processos. A complexidade do processamento paralelo não mais será escon-

dido no hardware com o paralelismo no nível de instruções (ILP) e técnicas de pipelines, assim como com projetos superscalares. Esse paralelismo deve ser abordado pelo software.

(b) Os processadores de uso geral (COTS - *commodity off-the-shelf*), por razões econômicas, tem sido usados em sistemas de alto desempenho. Porém, a arquitetura desses processadores *multicores* de propósito geral nem sempre é capaz de atender os requisitos das aplicações da pesquisa de ponta, mesmo quando o software é devidamente modificado. Então, além das diferenças dos *multicores* e dos multiprocessadores, os *multicores* podem incorporar à sua configuração elementos processadores de propósito especial, como os aceleradores de hardware, GPUs (*Graphics Processing Unit*) e FPGAs (*Field-Programmable Gate Array*). Os processadores *multicores* comercializados atualmente possuem diferentes projetos e ainda com essas várias opções de configurações de hardware dificultam o surgimento de uma arquitetura comum a partir da qual se desenvolva um modelo único de programação.

6. MacCool (2008) constata que a extração automática de paralelismo chegou ao ponto em que os ganhos obtidos são insignificantes, o que leva a um interesse renovado nos modelos de programação com paralelismo explícito. Em geral, espera-se que o modelo de programação expresse os aspectos mais importantes do modelo de processamento e a implementação da linguagem de programação simplesmente automatiza o mapeamento das computação desejada para a máquina alvo. Dessa forma, a linguagem de programação pode auxiliar o programador escondendo os detalhes desnecessários e aumentando a produtividade. Entretanto, os modelos de programação das linguagens de programação tradicionais são voltados para a programação sequencial e não expõem o paralelismo e o acesso à memória, dois aspectos importantes para o desenvolvimento de um sistema de alto desempenho.

7. Schepke e Maillard (2008) abordam a programação em ambientes computacionais com múltiplos níveis de paralelismo. Apresentam os principais recursos de programação utilizados em cada nível de paralelismo e propostas que exploram a programação em multiníveis, assim como as linguagens de programação: *Unified Parallel C* (UPC), *Co-Array Fortran* (CAF), Titanium, Chapel, Fortress e X10.

8. Speyer et al. (2008) alegam que da mesma forma que as linguagens de programa-

ção de alto nível substituíram as linguagens de baixo nível trocando as perdas de desempenho pelo aumento da produtividade, na programação para sistemas computacionais de alto desempenho deverá sacrificar o desempenho por ganhos de produtividade. Vários paradigmas de programação paralela se propuseram a criar este alto nível de abstração, como a linguagens Partitioned Global Address Space (PGAS), que incluem Co-Array FORTRAN (CAF), Unified Parallel C (UPC), e Titanium (uma derivação do Java). Todas possuem um consórcio de desenvolvedores por trás delas. Outras novas linguagens estão sendo desenvolvidas como parte do DARPA High Productivity Computing Systems (HPCS), que inclui X10 (IBM), Chapel (Cray), and Fortress (SUN). Todas essas linguagens tentam esconder a complexidade da programação por troca de mensagens. O trabalho se propõe a avaliar essas novas linguagens em três etapas. Apresenta o resultado da primeira etapa que corresponde à revisão de literatura. A segunda etapa pretende avaliar as linguagens através da implementação de aplicações conhecidas, e a terceira e última etapa envolve um grupo de alunos que irá desenvolver programas em cada uma das linguagens avaliadas. A segunda e terceira etapas estão ainda em andamento.

B.4.2 Sistemas Operacionais

Sistemas operacionais tem uma forte influência no desempenho da execução de uma aplicação, sendo responsabilidade desses a gerência dos recursos de hardware utilizados pelas aplicações, como o gerenciamento da memória e da execução de processos. Os sistemas operacionais comerciais atuais foram desenvolvidos para computadores com suporte a memória compartilhada, e não atendem às várias inovações que estão surgindo no projeto dos processadores *multicores*. Para manter a escalabilidade do desempenho com o aumento do número de cores por processador e a compatibilidade com versões anteriores, os sistemas operacionais atuais estão tendo um árduo trabalho. Segundo Baumann et al. (2009), a recente atualização do Windows7 chegou a 6000 linhas de código em 58 arquivos.

Em (Boyd-Wickizer et al., 2010), os autores verificam a escalabilidade de sete aplicações no Linux kernel (2.6.35-rc5, released July 12, 2010) em um computador com 48 cores. Os gargalos detectados que não puderam ser resolvidos com modificação no código das aplicações levaram a introdução de uma técnica chamada de *sloppy counters*

que corresponde a 3002 linhas de código modificado no Linux kernel (patched kernel PK). Concluem que ainda não há razão para se abandonar o uso do Linux atual, porém não estende a conclusão para além de 48 cores.

Em (Hillar, 2009) é feita uma comparação dos sistemas operacionais mais populares em relação aos processadores *multicores*:

- MacOS: possui uma vantagem sobre os concorrentes por rodar somente em hardware certificado e a mesma companhia que desenvolve o hardware também desenvolve o software. Roda sobre processadores Intel, faz uso de vetorização e do modelo SIMD (single instruction multiple data) na sua nova versão, o Mac OS X Snow Leopard. Introduz o Grand Central Dispatch (GCD) que faz a alocação de tarefas por core. GCD usa tarefa como uma abstração de threads para esconder detalhes do programador.

- FreeBSB: sistema operacional *free* e *open source*, trabalha bem tanto com multiprocessadores quanto *multicores*.

- Linux: possui versão otimizada para trabalhar com *multicore*. Tanto FreeBSB quanto Linux trabalham bem com arquiteturas NUMA, porém quanto maior o número de cores mais otimizações são necessárias para se ter um escalonador eficiente.

- Windows 2008 Server R2 e Windows 7 atendem até 256 cores lógicos e as versões 64-bits oferecem funcionalidades para se aproveitar das vantagens oferecidas pela arquitetura NUMA.

Conclui-se no trabalho de (Hillar, 2009), que não basta os sistemas operacionais oferecerem funcionalidades para maximizar o uso dos processadores *multicores*, as aplicações tem que fazer uso das mesmas para que o seu desempenho melhore. O autor também sugere que vai se chegar a um ponto em que não será mais possível que os novos sistemas ofereçam a compatibilidade com sistemas anteriores.

Baumann et al. (2009) afirmam que os processadores *multicores* irão ficar cada vez mais parecidos com redes de computadores e que devemos utilizar as ideias que surgiram e são utilizadas nos sistemas distribuídos. Propõem uma nova estrutura de sistema operacional que trata as máquinas *multicore* como uma rede de cores independentes - *multikernel*, sem compartilhamento de memória. Apresenta as usuais funcionalidades

do sistema operacional como um sistema distribuído, onde processos se comunicam por troca de mensagens. A arquitetura do sistema operacional *multikernel* se propõe a ser escalonável com aumento do número de cores e através da sua estrutura modular suas funcionalidades podem ser utilizadas em diferentes arquiteturas *multicores*. Introduz o Barrelfish, um implementação do modelo *multikernel* construído a partir da colaboração do ETH Zurich e Microsoft Research Cambridge na Inglaterra. Encontra-se disponível para download em `http://www.barrelfish.org/`.

Em (Boyd-Wickizer et al., 2008), é mostrado que o desempenho das aplicações *multicore* pode ser limitado pelo sistema operacional quando a aplicação usa o sistema operacional com frequência e os serviços do sistema operacional usam estruturas de dados compartilhadas e modificadas por múltiplos cores. Quando a aplicação não precisa desse compartilhamento, o sistema operacional pode se tornar um gargalo. Propõe a inclusão no sistema operacional de abstrações que permitirão às aplicações controlar o compartilhamento entre cores e usufruir da vantagem de se ter número abundante de cores para se dedicarem a funções específicas do sistema operacional.

O sistema operacional *Factored operating systems (fos)* é apresentado como um sistema operacional escalável para *multicores* é apresentado em (Wentzlaff e Agarwal, 2009). *Factored operating systems (fos)* prevê que se o número de cores nas novas gerações de processadores vai dobrar a cada 18 meses, a abordagem atual de se otimizar os sistemas operacionais atuais não terá mais efeito pois a taxa de aumento de paralelismo ultrapassara a taxa que os projetistas de sistemas operacionais serão capazes de redefinir os sistemas. Apresenta os problemas de escalabilidade com o aumento de cores dos sistemas operacionais atuais e então propõe o novo sistema operacional: *factored operating system - fos*. Tendo a escalabilidade como principal requisito de projeto e objetivando sistemas com *manycores* (mais de 1000 cores), *fos* é construído com base na troca de mensagens e inspirado na forma como a Internet provê serviços.

Em (Betti et al., 2008) é apresentada uma modificação do Linux, o ASMP-Linux, como um sistema operacional em tempo real para sistemas *multicore* com alto desempenho, e sem a necessidade das aplicações atuais serem recompiladas ou modificadas. Foca os sistemas embarcados e os requisitos de desempenho em tempo real. Mostra resultados encorajadores de *benchmarks* executados em diferentes plataformas de hardware e diferentes configurações. O ASMP-Linux é um *patch* para Linux kernel 2.6.21.

B.4.3 Ensino

O ensino de arquiteturas e programação paralela não é contemplado na maioria das grades atuais dos cursos básicos de formação de profissionais da área de desenvolvimento de software. Trabalhos como o de Marowka (2008) que apresenta uma estrutura de curso de programação paralela implementa em Israel, no *Shenkar College of Engineering and Design*. Outro trabalho é o de Freitas et al. (2008) que propõe um ambiente de simulação para o ensino de arquiteturas de processadores *multcore*.