Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Informática
Curso de Bacharelado em Informática

Estudo comparativo da eficiência de um Algoritmo Genético Sequencial frente um Algoritmo Genético Paralelo

Osmar dos Santos

CASCAVEL
2009

Osmar dos Santos

Estudo comparativo da eficiência de um Algoritmo Genético Sequencial frente um Algoritmo Genético Paralelo

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Msc. André Luiz Brun

CASCAVEL
2009

Osmar dos Santos

Estudo comparativo da eficiência de um Algoritmo Genético Sequencial frente um Algoritmo Genético Paralelo

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Informática, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Msc. André Luiz Brun (Orientador)
Colegiado de Informática, UNIOESTE

Prof. Msc. Guilherme Galante (Co-Orientador)
Colegiado de Informática, UNIOESTE

Prof. Dr. Adair Santa Catarina
Colegiado de Informática, UNIOESTE

Cascavel, 10 de dezembro de 2009

"... life’s more painless for the brainless..."

Stephen Schwartz

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

Neste trabalho foram estudadas técnicas de paralelismo e algoritmos genéticos para o desenvolvimento e análise de bibliotecas genéricas capazes de resolver problemas com o uso de algoritmos genéticos, de forma sequencial e paralela em dois modelos. Na versão sequencial foi utilizado o modelo clássico e as arquiteturas paralelas escolhidas basearam-se em uma abordagem simples, onde diversas instâncias sequenciais são executadas paralelamente e ao fim são recolhidos os resultados, e uma híbrida que envolve os modelos mestre-escravo e de multi população. Para avaliar as soluções implementada utilizou-se o problema do caixeiro viajante, onde o desempenho, em relação ao tempo de execução, e a qualidade das soluções obtidas pelas diversas versões foram confrontadas. Observou-se que as versões paralelas apresentam tempo de execução menor do que a serial, com a paralela simples apresentando-se mais eficiente, e que os resultados obtidos pela versão paralela híbrida são melhores aos do método sequencial.

Palavras-chave: Algoritmo Genético, Paralelismo, Caixeiro Viajante, Estrutura Genérica.

Capítulo 1

Introdução

Algoritmos genéticos (AGs) são utilizados na resolução de problemas de otimização e busca. Embora ele não garanta soluções exatas, elas são consideradas aceitáveis pela proximidade do valor ótimo.

O processo de execução de um algoritmo genético envolve diversas etapas e processos, o que para condições onde se tenha um grande espaço de busca ou muitas gerações são necessárias para se encontrar um solução aceitável, pode consumir tempo considerável, cenário onde o uso de paralelismo pode ser uma melhor opção.

1.1 Objetivos

O paralelismo intrínseco dos AGs, onde os indivíduos são avaliados de modo independente, motivou a realização deste estudo, onde se buscou avaliar estrategias de paralelização computacional dos AGs.

Devido a isso, este trabalho tem como objetivo secundário o de desenvolver bibliotecas genéricas que facilitem o uso de algoritmos genéticos para a resolução de problemas de forma sequencial e paralela.

A partir do desenvolvimento destas bibliotecas foi possível verificar se o uso do paralelismo resultará em melhoramento do desempenho, em relação ao tempo de execução, e na qualidade das solução.

Para a versão sequencial da biblioteca é utilizada uma adaptação do algoritmo genético clássico. Já para a versão paralela foram desenvolvidas duas implementações, estas utilizando-se de arquiteturas de paralelismo diferentes.

Os modelos de paralelismo escolhidos foram o trivial, onde tem-se instâncias diferente do mesmo algoritmo sequencial trabalhando sobre conjuntos de dados diferentes, e o híbrido, onde duas outras abordagens, a mestre-escravo e a multi-população, são utilizadas. O objetivo de se utilizar dessas duas abordagens é o de comparar qual apresenta melhores resultados (em termos de qualidade e complexidade temporal) para o estudo de caso que será proposto.

Comparou-se o desempenho das abordagens utilizando-se os fatores de tempo de execução do algoritmo e qualidade das soluções obtidas (adaptabilidade do melhor indivíduo ao ambiente).

Como base para o estudo comparativo (ou grupo controle) utilizou-se a versão sequencial, possibilitando inferir se um algoritmo genético paralelo preserva a qualidade dos resultados apresentando um tempo de execução menor.

Para o estudo de caso foi escolhido o problema do caixeiro viajante. Tal opção ocorreu por este fazer parte da gama de problemas de otimização e busca mais difícil.

1.2 Organização do trabalho

O trabalho foi assim estruturado: no capítulo 2 apresentam-se os conceitos relacionados à paralelismo e suas arquiteturas, os quais são utilizados como base para o melhor entendimento do algoritmo genético paralelo.

No capítulo 3 foi estudado o que são algoritmos genéticos e seu comportamento clássico, assim como alguns métodos de cruzamento e mutação baseados na codificação real.

No capítulo 4 os conhecimentos estabelecidos nos capítulos anteriores são utilizados para o estudo dos algoritmos genéticos paralelos, onde são apresentadas algumas arquiteturas disponíveis, dentre as quais estão as duas escolhidas para implementação deste trabalho.

No capítulo 5 tem-se a metodologia deste trabalho, onde é apresentado como se deu seu desenvolvimento e os motivos que levaram às escolhas feitas, assim como suas peculiaridades.

No capítulo 6 são apresentados os resultados obtidos, assim como a análise, quanto ao desempenho, tempo de execução do algoritmo, e qualidade das soluções.

No Apêndice A são apresentadas as bibliotecas desenvolvidas e como ás utilizar para a resolução de outros problemas.

Capítulo 2

Paralelismo

Neste capítulo é apresentada uma breve revisão acerca de paralelismo e sistemas distribuídos, com objetivo de demostrar os tipos de paralelismo e arquiteturas, dando ênfase à arquitetura de memória distribuída.

A escolha pela arquitetura de memória distribuída se justifica pelo tipo de ambiente disponível para a realização dos testes, o qual é um cluster, conjunto de computadores com memória própria e que cooperam entre si para a realização de uma tarefa.

Ao fim são mostradas as métricas utilizadas para verificar o desempenho do algoritmo paralelo desenvolvido, em relação ao seu equivalente sequencial, sendo elas o *speedup* e a eficiência.

2.1 Introdução

Paralelismo é a divisão de um problema ou tarefa de forma que a mesma possa ser executada de forma concorrente, várias partes ao mesmo tempo, e em processadores diferentes, com o objetivo de se chegar a uma solução de modo mais rápido ou eficiente.

Com o paralelismo é possível proporcionar o aumento do poder computacional de forma barata, pelo uso numeroso de processadores de baixo desempenho trabalhando em conjunto, perfazendo uma capacidade de processamento elevada. Ou ainda, utilizando-se de hardware desenvolvido especificamente para este objetivo.

Ao se paralelizar um programa o objetivo principal é manter todos os processadores ocupados, de forma balanceada, e ao mesmo tempo manter o nível de comunicação entre os mesmo condizentes com o meio de comunicação utilizado. Geralmente esse é o ponto crítico de um processamento paralelo [16].

A ideia principal na paralelização de um programa é dividir o problema entre os diversos processadores disponíveis, o que pode ser feito de modo funcional ou de dados [16].

2.2 Arquitetura

A arquitetura de um sistema paralelo diz respeito à maneira como as instruções são executadas, de forma unitária ou múltipla, e como os dados fluem pelo sistema, daí vem a definição de Flynn [19], que as trata como fluxo de dados (*data stream*) e fluxo de instruções (*instruction stream*).

A arquitetura paralela pode ser dividida em SIMD e MIMD [19]; esta última ainda pode ser subdividida em outros três tipos de arquitetura MIMD, isso de acordo com o tipo de organização de memória.

Ao tratar sobre arquitetura MIMD será feita uma explanação acerca dos tipos de organização de memória presentes na mesma, dando enfoque à memória distribuída, tendo-se em vista o ambiente para desenvolvimento deste trabalho, o qual segue estas características.

2.2.1 SIMD

A arquitetura SIMD (*Single Instruction Multiple Data*) é formada de inúmeros processadores simples, cada qual com uma memória local onde os dados a serem trabalhados são armazenados. Uma mesma instrução é executada de forma simultânea por todos os processadores em seus dados locais e seguem o comando de um controlador. Os processadores podem se comunicar uns com os outros para realizar operações de deslocamento (*shifts*) e outras operações de *array*. Também são conhecidos como sistemas de processamento vetorial, [21] [8] [41].

A Figura 2.1 exemplifica como é a arquitetura SIMD.

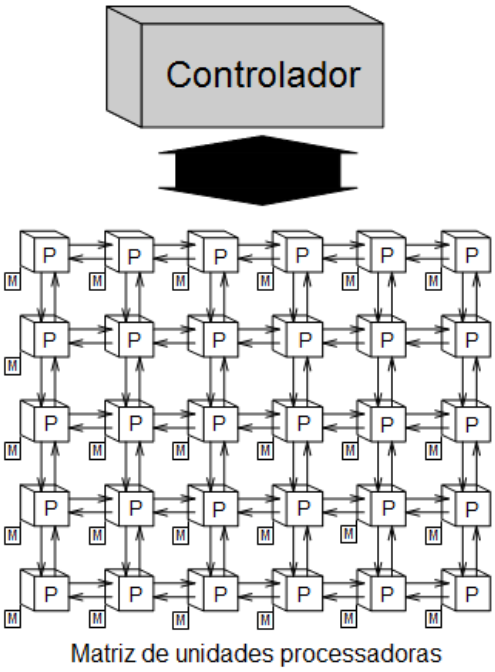

Figura 2.1: Exemplo da arquitetura SIMD.
Fonte: Adaptado de *Figure 1: An example of a SIMD architecture* em [8].

2.2.2 MIMD

A arquitetura MIMD (*Multiple Instructions Multiple Data*), é formada de inúmeros processadores independentes. Estes capazes de executar instruções diferentes uns dos outros, em seus dados locais. Esse tipo de arquitetura fornece uma grande flexibilidade na execução de algoritmos paralelos, diferentemente da arquitetura SIMD onde o processamento tem propósitos específicos, além de apresentarem um bom desempenho em virtude de seus elementos realizarem processamento assíncrono, independentes entre si [1].

Esta arquitetura é subdividida de acordo com o tipo de organização de memória, que pode ser memória compartilhada, memória distribuída e memória virtual compartilhada.

Memória compartilhada

Neste tipo de organização, também conhecidos como multiprocessadores, existem diversos processadores e todos tem acesso à uma unidade de memória global, através de algum meio de interconexão ou barramento. A comunicação entre os processadores é feita através da leitura e

escrita de dados nesta memória. Esse tipo de comunicação faz com que o tempo de acesso a qualquer fragmento de informação seja o mesmo.

Essa arquitetura mostra-se vantajosa pela facilidade de se programar a mesma, uma vez que a comunicação é implícita através da memória. Porém isso faz com que seja necessária a adoção de uma técnica de controle de acesso à memória, como por exemplo os semáforos.

Seu ponto fraco está na falta de escalabilidade uma vez que, na ocorrência de muitos acessos concorrentes à memória, haverá um gargalo devido ao uso do mesmo barramento por todos os processadores. Uma solução para este problema é o uso de *caches*, as quais devem ter sua consistência garantida por alguns tipo de mecanismo de coerência de cache, implementado em hardware ou software [21] [8] [41] [1].

A Figura 2.2 mostra a arquitetura de memória compartilhada, onde tem-se um bloco enorme de memória ligado a um barramento e vários processadores ligados a este.

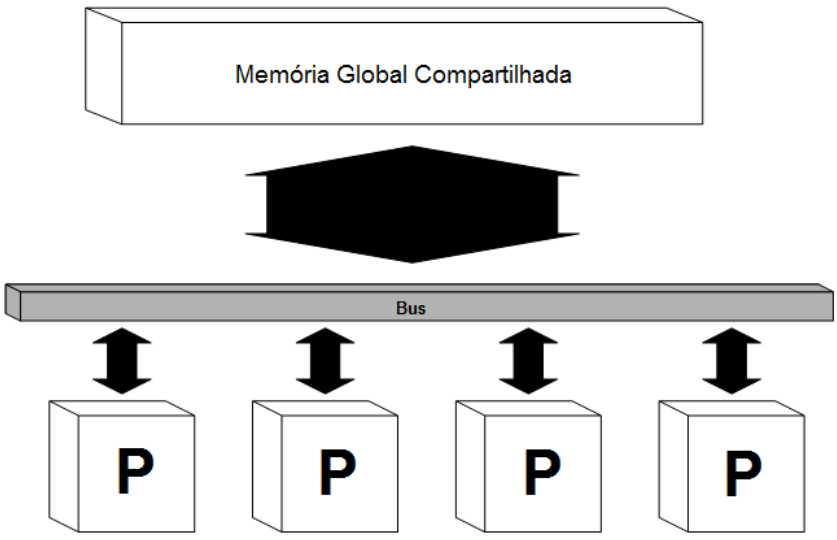

Figura 2.2: Exemplo memória compartilhada.
Fonte: Adaptado de *Figure 2: A example of a shared memory architecture* em [8].

Memória distribuída

Neste tipo de organização, também conhecido como multicomputadores ou sistemas de troca de mensagens (*message passing systems*), existem diversos processadores e cada um tem sua própria quantidade de memória. O acesso a memória é permitido somente para a memória local, na necessidade de acesso à dados de um outro processador deve-se enviar uma mensagem

com o pedido dos dados a este, que por sua vez responderá com uma outra mensagem contendo os dados requisitados.

Um ponto relevante nesse tipo de organização está no fato de que quanto maior a distância entre os processadores maior será o tempo de acesso a algum dado da memória remota. Este tempo é denominado latência [55].

Devido às suas características, esse tipo de organização pode ser implementado utilizando-se de um cluster, computadores tradicionais interconectados por um meio de comunicação.

A Figura 2.3 mostra a arquitetura de memória distribuída, onde se tem vários computadores conectados em rede para realizar uma tarefa em comum.

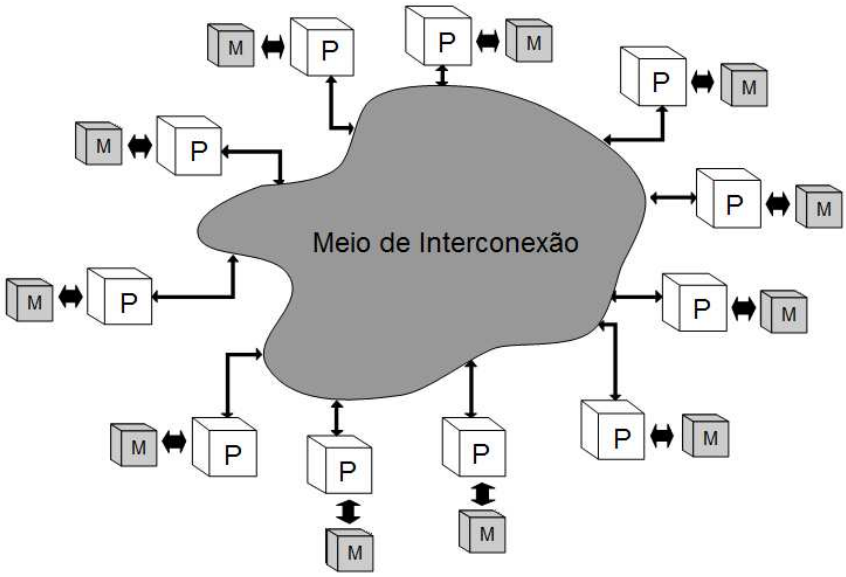

Figura 2.3: Exemplo memória distribuída.
Fonte: Adaptado de *Figure 3: A example of a distributed memory architecture* em [8].

Memória compartilhada virtual

Nesse tipo de organização de memória, também conhecido como DSM (*Distributed Shared Memory*), tenta-se unir as qualidades e benefícios provenientes das organizações de memória distribuída - facilidade de se construir e boa escalabilidade - do ponto de vista do projetista e de memória compartilhada - comunicação entre processos através da memória - do ponto de vista do programador, produzindo uma arquitetura que seja fácil de projetar e programar. Onde usa-se uma plataforma de memória distribuída, e através de mecanismos implementados em

hardware e software um ambiente de memória compartilhada é emulado [41] [34] [48].

2.2.3 Cluster

Um cluster é um sistema distribuído, constituído de dois ou mais computadores interconectados em rede, os quais trabalham em conjunto para realizar uma determinada tarefa ou tarefas.

Embora possa ser constituído de sistemas homogêneos, na sua maioria são computadores heterogêneos, com características distintas quanto a [5]:

- Arquitetura - pode-se estar fazendo uso de computadores com arquiteturas diferentes como por exemplo: CISC (x86), RISC (SPARC);
- O formato de dados - geralmente arquiteturas diferentes apresentam formato de dados diferentes, e mensagens trocadas devem ser entendidas por ambos os lados. O sistema de troca de mensagens deve manipular estes dados de forma a fazer com que a comunicação possa ocorrer;
- A potência computacional - o uso de estações distintas deve ser observado, uma vez que computadores mais velozes não devem ficar esperando por outro mais lentos, ou seja, é necessário um balanceamento de carga;
- Carga de trabalho - por possibilitar o uso de estações e rede comuns com usuário, isso pode afetar negativamente o desempenho de um sistema que explora o paralelismo.

Mesmo apresentando limitações, devido a sua heterogeneidade, sistemas distribuídos oferecem benefícios como: custo reduzido, desempenho alto, exploração da heterogeneidade para resolução de alguns tipos de problema e acesso a um ambiente conhecido.

Seu uso se justifica pelo baixo custo e escalabilidade, onde a necessidade por aumento de poder computacional pode ser conseguida com a adição de mais processadores. Enquanto que sistemas desenvolvidos especialmente para alto desempenho apresentam um grande custo e a melhora de performance só é conseguida com a atualização do sistema como um todo.

2.3 Troca de mensagens

A troca de mensagens, ou passagem de mensagem (*Message Passing*), é um modelo computacional que conceitualiza operações entre programas. Nela o ponto principal é que os processos se sincronizem e realizem comunicação através da troca de mensagens [41] [1] [2] [29] [39] [45]. Do ponto de vista dos processos uma troca de mensagem é apenas uma chamada à uma interface de troca de mensagens, a qual é responsável de tratar a ligação física que junta os processos ou processadores.

Esse modelo pode ser usado para comunicação de processos com memória compartilhada ou distribuída, desde que ambos sejam baseados no envio e recebimento de mensagens. Para a efetivação de uma operação de troca de mensagem é necessária a cooperação entre os processos envolvidos.

No modelo de programação com troca de mensagens o elemento central é a mensagem, a qual é trocada entre os processos. A troca de mensagem serve como um meio para a cópia de dados da memória de um processo para a memória de outro processo, assim como para a sincronização entre os mesmos. Além do dado, propriamente dito, uma mensagem contém um cabeçalho com informações importantes, estas utilizadas pelo sistema de troca de mensagens para poder realizar todo o processo.

De acordo com Booth [8], as seguintes informações devem estar contida em um cabeçalho, algumas das quais podem ser disponibilizadas para o processo que as recebe, por meio do sistema de troca de mensagens:

- Qual processo esta enviando a mensagem;
- Qual a localização dos dados no processo que esta realizando o envio;
- Que tipo de dado está sendo enviado e em qual quantidade;
- Qual(is) processo(s) esta(ão) recebendo a mensagem;
- Onde os dados devem ser colocados no processo que os está recebendo;
- A quantidade de dados que o processo que recebe está preparado para aceitar.

O sincronismo entre os processos, é obtido por meio das funções de requisição de mensagens (*message querying functions*), onde o sistema de troca de mensagens é responsável por fornecer alguns informações sobre o progresso da comunicação, permitindo que um processo recebedor esteja consciente sobre o recebimento de mensagens, assim como um processo realizando um envio possa descobrir se sua mensagem foi entregue ou não.

2.3.1 Troca de mensagens em sistemas paralelos

Programação paralela é a cooperação entre processos com o intuito de resolver um problema em comum. Existem dois pontos importantes quando se está programando processos que cooperam entre si. Primeiramente deve ser definido que processos serão executados pelos processadores, assim como especificar como estes processos realizam sua sincronização e troca de dados.

Com relação a troca de mensagens e sincronismo, é preciso que os processos que irão se comunicar tenham acesso ao sistema de comunicação. A interface de comunicação deve providenciar algum tipo de mecanismo afim de especificar quais processos fazem parte da comunicação. E por fim o projetista precisa de algum meio para relacionar o término de uma chamada a um procedimento (modo como o modelo de troca de mensagens trabalha) com o real término da comunicação.

Com base nisso são definidos alguns componentes de comunicação que um sistema de troca de mensagens deve disponibilizar; componentes necessários para o registro de processos e o início e término do processo de troca de dados em um sistema paralelo:

- Acesso - os processos devem ser registrados junto à interface de troca de mensagens, com isso outros processos podem saber onde o mesmo se encontra e como o acessar. No modelo de computação MIMD, a interface de comunicação oferece alguns mecanismos de facilitação, os quais permitem que processos sejam iniciados e registrados no meio do processo de computação, diferentemente do modelo SIMD, onde todos os processos devem ser registrados antes da computação começar;

- Endereçamento - deve ser possível endereçar as mensagens, isto é, estas devem conter informações relevantes que auxiliem a sua entrega assim como sua devolução;

- Recepção - o processo que está recebendo a mensagem deve ser capaz de lidar com os dados que estão chegando, se por um acaso o processo for incapaz de tratar tais dados, como no caso de *buffer* insuficiente, efeitos adversos podem ocorrer, como mensagens truncadas ou até mesmo descartadas pelo sistema de troca de mensagens. Para o problema específico do *buffer*, alguns sistemas permitem que o mesmo seja controlado pelo processo; outros deixam tal controle transparente para o projetista. Ainda, a correta execução de um programa paralelo baseia-se na capacidade do processo, realizando o envio das mensagens, de identificar falhas na entrega dos dados.

2.3.2 MPI (Biblioteca de troca de mensagens)

O MPI (*Message Passing Interface*) é na verdade um padrão para ambientes de passagem de mensagens, computadores MIMD com memória distribuída, com o objetivo principal de padronização, garantindo a portabilidade de aplicações que fazem uso de troca de mensagens entre diversas arquiteturas de computadores.

O comitê responsável pela especificação da MPI, é composto por representantes de aproximadamente 40 organizações, o que inclui a maioria dos fabricantes de máquinas com MPP (*Massively Parallel Processing*, processamento maciçamente paralelo), além de universidades e laboratórios envolvidos na computação paralela [34] [53].

Na MPI toda paralelização é explicita, ou seja, o projetista é responsável por fazer-lá utilizando-se das funções disponíveis. É possível encontrar definição para funções que realizam [21] [11].

- Comunicação ponto a ponto: constituem o núcleo do MPI e são utilizados pra o envio e recebimento de mensagens entre dois processos;

- Comunicação coletiva (de grupo): utilizadas para comunicação com um grupo de processos, *broadcast*, e para a sincronização dos mesmo;

- Grupos de processos;

- Gerenciamento de processos: utilizadas para inicialização e finalização de processos, determinar o número de processos e identifica-los.

2.4 Tipos de paralelismo

Ao buscar dividir um programa sequenciais em partes para poder executá-lo de forma paralela, observa partes estritamente sequências, as quais não podem ser paralelizadas, e partes paralelizáveis. Embora o objetivo desta divisão de trabalho é ganhar tempo e performance, em alguns casos ela pode gerar uma sobrecarga, como na troca de mensagens, onde ele é gerado pela comunicação adicional entre os processos.

Ao se verificar o desempenho final de um programa que fora paralelizado é preciso levar em conta essa sobrecarga. O ganho deve ser maior que o mesmo, ou a paralelização é considerada ineficiente.

Por isso, faz-se importante o estudo de algumas técnicas de decomposição do problema, afim de auxiliar a descoberta da mais adequada para a situação abordada no ambiente disponível. A seguir são mostrados algumas técnicas de decomposição do problema.

2.4.1 Decomposição trivial

É o modo mais fácil de decomposição, embora na verdade não se tenha a decomposição do problema. Esta técnica pode ser aplicada a problemas onde um programa deve ser executado, de forma independente, em várias entradas diferentes. Com isso a introdução do paralelismo está na execução de várias instâncias do mesmo programa sequencial de forma paralela. Como não se tem comunicação entre os processos é ideal para sistemas com baixa capacidade de comunicação [8].

2.4.2 Decomposição funcional

Na decomposição funcional o programa é dividido em vários subprogramas. Sua forma mais simples se apresenta no modo de um pipeline, onde cada unidade de dado passa pelos subprogramas de forma sequencial. O paralelismo esta no fato de se ter vários conjuntos de dados movendo-se através do pipeline de forma simultânea. O nível de paralelismo nesse modelo esta limitado pelo número de *pipelines* disponíveis e a eficiência está ligada diretamente com a capacidade de cada estágio executar suas tarefas em um período de tempo igual aos demais, ou seja, o balanceamento do pipeline. Ainda, a decomposição funcional é extremamente dependente do problema, além de ter o paralelismo limitado pelo programa [8].

2.4.3 Decomposição de dados

Consiste na divisão dos dados a serem trabalhados, de forma igual, entre todos os processadores, estes executando o mesmo programa, sequencial, completo, porém em um subconjunto dos dados [8]. Sua forma mais simples se apresenta no modelo Mestre-Escravo, onde se tem uma entidade principal, denominada Mestre, e subunidades denominadas Escravos. O Mestre é responsável por decompor o conjunto de dados e repassa-lós aos Escravos, os quais realizam algum tipo de cálculo e devolvem o resultado.

2.4.4 Granularidade

Ao se escolher ou definir um tipo de paralelismo a ser usado um ponto importante é a granularidade que será empregada. Pois essa é o nível de paralelismo que o problema apresenta e está relacionado ao tamanho das unidades de trabalho que são submetidas aos processadores.

A granularidade de um programa está ligada com o tipo de plataforma empregada, podendo ser subdividida em [30] [38]:

- Granulação grossa - onde o paralelismo se apresenta em termos de processos e programas. Estes tendem a executar um grande número de instruções entre os pontos de sincronização. Embora apresente uma facilidade para se aumentar a performance do sistema, tem-se uma dificuldade em se obter um balanceamento de cargas eficiente. O nível de computação é alto e o de comunicação é baixo; sistemas distribuídos se encontram nessa definição [11] [41];

- Granulação fina - onde o paralelismo se apresenta baseada em instruções ou operações. Programas executam um pequeno número de instruções entre os pontos de sincronização. Embora se tenha uma facilidade em obter um balanceamento de cargas eficiente, estes sistemas implicam em um alto nível de comunicação, podendo haver mais comunicação do que computação, diminuindo a performance do mesmo. Usado geralmente em plataformas com um grande número de processadores, estes pequenos e simples [11] [41];

- Granulação média - esta granulação se situa entre a grossa e a fina, onde o paralelismo se apresenta em termos de procedimentos.

2.5 Medida de desempenho

Assim como definido, o objetivo principal da aplicação de um algoritmo paralelo é o de se obter um desempenho melhor, comparando-se com a solução sequencial para o mesmo problema.

Com isso, é necessário estudar algumas métricas para realizar a comparação e averiguação da qualidade do algoritmo empregado. Porém, antes é preciso observar uma regra muito importante que rege tal estudo, a lei de Amdahl (*Amdahl's Law*) [41].

A lei de Gene Amdahl, de 1967, diz que a limitação do tempo de execução de um programa paralelo está relacionado ao tempo de execução de sua parte sequencial. Um programa paralelo, que tem 20% do seu código seqüencial e 80% paralelo tem o tempo mínimo de execução igual a 20% do tempo do mesmo programa seqüencial.

Essa lei fez o surgimento da ideia de paralelismo de trechos de computação intensiva dos programas, e não partes do programa em si [8].

Além da lei de Amdahl, outros fatores como sobrecarga de comunicação, nível de paralelismo e/ou balanceamento de cargas utilizados incorretamente, prejudicam o desempenho final de um programa paralelo.

2.5.1 *Speedup*

Speedup pode ser visto como o aumento de velocidade, ou diminuição do tempo, de execução de um determinado processo em n processadores, em relação a execução deste mesmo processo em apenas 1 processador [39] [30], como pode ser observado na Equação 2.1, onde T_1 é o tempo de execução com 1 processador e T_n é o tempo de execução com n processadores.

$$speedup = \frac{T_1}{T_n} \tag{2.1}$$

2.5.2 Eficiência

A medida de eficiência trata da relação entre o número de processadores utilizados e o *speedup* obtido, como observado na Equação 2.2, onde n é o número de processos concorrentes.

$$e = \frac{speedup}{n} \tag{2.2}$$

Capítulo 3

Algoritmos genéticos

Neste capítulo são revisados os conceitos de algoritmos genéticos que são necessários para o prosseguimento deste trabalho. Esses conhecimentos foram utilizados para a implementação da versão sequencial da biblioteca e como base para o entendimento de algoritmos genéticos paralelos.

3.1 Introdução

Algoritmos genéticos (AG's) inserem-se na computação evolutiva. É uma área de inteligência artificial (IA) que abrange as seguintes áreas de acordo com Bäck [4]: biologia, inteligência artificial, otimização numérica e engenharia, e permitem a simulação de processos de evolução natural. Definida por Bittencourt [7], como o desenvolvimento de computações artificiais inspiradas nas iterações biológicas realizadas pelos seres vivos para poderem viver e se reproduzir.

Eles são uma família de modelos computacionais inspirados na teoria evolucionária definida por Charles Darwin [12] e desenvolvidos por John Holland [27] na década de 60, simulando a reprodução sexuada da biologia [25].

De acordo com a teoria de Darwin [12], indivíduos ruins de uma população não sobrevivem, pois os mesmos são eliminados durante o processo de seleção. Já os indivíduos bons sobrevivem, de forma que são selecionados para o processo de recombinação e deste modo contribuem para a geração de melhores elementos.

A Figura 3.1 apresenta um exemplo clássico de seleção natural entre as girafas, onde a evolução permitiu que apenas os indivíduos com pescoços maiores sobrevivessem.

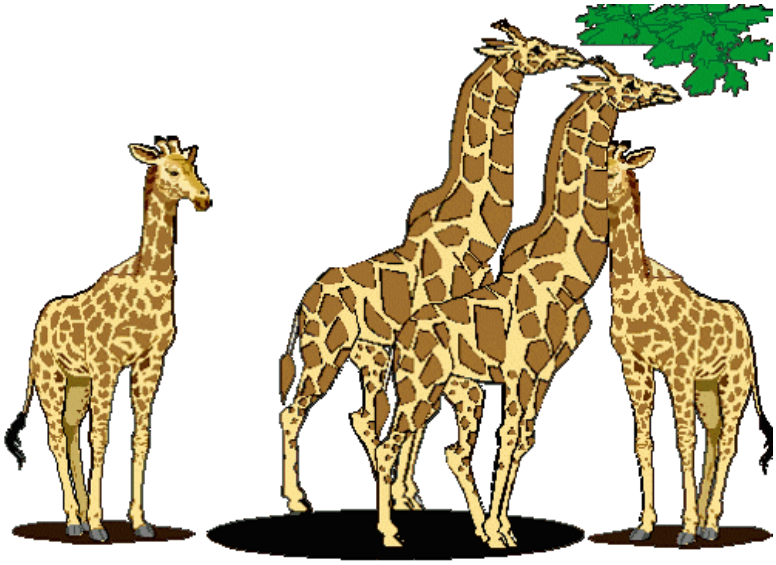

Figura 3.1: Evolução natural nas girafas [28].

Em sua pesquisa que originou os algoritmos genético Holland [27], tinha como centro o estudo formal do fenômeno de adaptação que ocorre na natureza e como este mecanismo de adaptação natural poderia ser portado para computadores [36].

No entanto, a sua popularização foi conseguida por David E. Goldberg [23], aluno de Holland, ao utilizar a técnica na resolução de um problema complexo de controle de transmissão de gás em duto.

Algoritmos genéticos podem ser definidos como método eficiente de otimização e busca, que utiliza-se de busca aleatória (aleatória pois a única informação presente é o valor da função objetivo), baseado em princípios evolucionários de populações de seres vivos [10] [6], onde através de um processo iterativo busca-se melhores cromossomos manipulando o seu conteúdo aleatoriamente. E durante o processo de reprodução ocorrem fenômenos atuantes no material genético dos cromossomos, como mutação e recombinação, resultando em uma população mais diversificada, a qual é submetida a processos de seleção natural, obtendo-se assim a sobrevivência dos mais aptos [23].

Goldberg [23] define algumas características de um algoritmo genético, as quais o diferencia de outros mecanismos normais de otimização e busca:

- Em um algoritmo genético a codificação do problema e a função de avaliação são dependentes do problema;

- A solução de um problema em específico é modelada em uma estrutura de dados semelhante a de um cromossomo, na qual são aplicados operadores de recombinação preservando informações críticas, tratando o conjunto de parâmetros de forma singular;
- Os resultados de um algorítimo genético são apresentados como uma população de soluções e não como uma solução única;
- Algoritmos genéticos não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado, fazendo o uso, somente, da função objetivo;
- São usadas transições probabilísticas, apenas, não fazendo o uso de regras determinísticas.

Ainda em uma comparação dos algoritmos genéticos com outros métodos de otimização e busca, Haupt [24] apresenta as seguintes vantagens:

- Pode realizar otimizações fazendo uso de parâmetros contínuos e discretos;
- Não necessita de informações secundárias;
- A busca é feita simultaneamente em um vasto espaço de busca;
- Pode ser feito o uso de um grande número de parâmetros;
- É adequado para uso em arquitetura de computadores paralelos;
- Tem capacidade para otimizar parâmetros complexos;
- Tem como resultado possíveis ótimas soluções, e não apenas uma;
- Faz uso de parâmetros codificados;
- Faz uso de dado gerados numericamente, experimentalmente e funções analíticas.

3.2 Algoritmo genético clássico

O algoritmo genético clássico opera sobre um conjunto de *n* indivíduos, chamado de população, onde cada indivíduo é composto por uma seqüência de bits, os quais representam uma solução em potencial para o problema em estudo.

Segundo Goldberg [23], a evolução geralmente se inicia a partir de uma população inicial de indivíduos de um conjunto de soluções criado aleatoriamente e é realizada por meio de gerações.

A cada geração, a adaptação, otimização, de cada indivíduo da população ao problema é avaliada. Essa quantificação é determinada pela qualidade de seu cromossomo como solução, e é obtida a partir da avaliação do mesmo pela função objetivo do problema, resultando em um valor chamado de fitness.

Essa avaliação é feita de modo que cromossomos que representem uma melhor solução tenham maiores chances de se reproduzirem em relação aos que representem um solução pior. Esta definição de solução melhor ou pior está relacionada à população atual.

Indivíduos são selecionados para fazer o papel de pais dos novos indivíduos da popularização baseando-se na sua qualidade relativa, uma qualidade em relação aos outros indivíduos da população.

O método utilizado para esta seleção é o método de Monte Carlo, onde as chances de um indivíduo ser sorteado é relativa a qualidade deste em relação a população. Os pais selecionados então sofrem o processo de cruzamento de forma randômica, formando assim os novos filhos. A recombinação entre dois indivíduos ocorre apenas se uma probabilidade de cruzamento for atingida.

Os filhos por sua vez sofrem um processo de mutação, o qual é realizado com uma probabilidade pequena, gerando uma mudança na informação carregada. A nova população então é utilizada como entrada para a próxima iteração do algoritmo.

Esta nova população pode, ainda, ser composta por indivíduos selecionados através do elitismo, promoção de alguns melhores elementos para a próxima geração.

Este processo é repetido até que não se tenha mais uma melhora significativa na qualidade das respostas obtidas, indivíduos, ou algum outro critério seja satisfeito.

Mitchell [36] e Davis [13] apresentam a seguinte estrutura como sendo a de um algoritmo genético clássico.

1. Iniciar uma população;
2. Calcular o *fitness* dos indivíduos da população;
3. Selecionar pares de indivíduos;
4. Recombinar os pares selecionados, afim de gerar os filhos;
5. Mutar os filhos;
6. Atualizar a população com os novos indivíduos;
7. Verificar se o ponto de parada foi encontrado, em caso negativo voltar para o passo 2.

Embora essa seja a abordagem clássica, muitas outras são usadas, onde se tem a codificação do cromossomo em outra forma que não a binária, uso de operadores de cruzamento e mutação mais complexos. E mesmo essas novas abordagens sendo tidas como algoritmos evolutivos também são algoritmos genéticos, devido ao fato de sua estrutura algorítmica ser a mesma de um algoritmo genético [35].

3.2.1 Codificação binária (clássica) versus codificação real

A etapa de escolha da codificação é tida como uma das mais importantes, pois esta deve ser capaz de armazenar informações precisas de uma solução, de forma que a computação com a mesma não seja comprometida em relação ao desempenho. O significado das informações carregadas por um indivíduo nunca é conhecido pelo mesmo [14].

Com isso, inicialmente Holland [27], ao definir algoritmo genético, propôs o uso de cadeias binárias para representar uma solução, chamada codificação binária ou clássica. Essa abordagem é possível através da discretização do domínio do problema.

A partir de 1989 com os trabalhos de Lucasius e Kateman [33] e Davis [13] a representação através de valores reais (não mais binários) passou a ser muito utilizado em problemas com espaço de busca muito grande, complexos e não completamente conhecidos, onde variáveis contínuas são empregadas [26].

Na codificação real o indivíduo é um vetor de números reais, onde a precisão do mesmo depende do computador onde está sendo executado o algoritmo. O tamanho do elemento é o mesmo do vetor de solução, deste modo, cada gene representa uma variável do problema. O valor dos genes são restritos ao intervalo estabelecido para a variável, pelo projetista, baseado no

domínio do problema, que ele representa, deste modo os operadores genéticos devem observar este pré-requisito.

De acordo com os estudos de Herrera, Lozano e Verdegay [26] e Santa Catarina [43], os melhores benefícios advindos do uso codificação real sobre a codificação binária está no fato de que o aumento do domínio do problema não implica em sacrificar a precisão da solução como nesta. Aquela ainda se mostra melhor em desempenho computacional, tempo, quando se usa cadeias grandes, já que ao fazer uso de cadeia binária é necessário realizar a conversão dos valores para se obter o valor *fitness* do indivíduo.

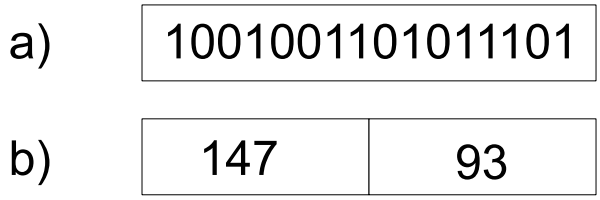

Figura 3.2: Exemplo codificação. a) representa a codificação binária e b) a real.

A seguir cada etapa do algoritmo genético, usando codificação real, é explicada com mais detalhe.

3.2.2 Inicialização

Inicialmente é gerada uma população de *n* indivíduos que, de acordo com Silva [44], pode ser obtida das seguintes formas:

- Aleatoriamente, onde são gerados genes de forma aleatória, porém dentro do intervalo de busca;
- Com indivíduos oriundos de outros processos, pode ser de outras iterações do algoritmo;
- De forma heurística ou controlada, quando se tem um conhecimento prévio sobre a região do espaço de busca ou onde a solução possa se encontrar, assim como o conhecimento de soluções aceitáveis;

Cada um destes indivíduos produzidos representam uma possível solução para o problema. De acordo com Tanomaru [49], a geração da população inicial deve assegurar a diversidade da população, ou seja, a cobertura abrangente do espaço de busca, de modo a garantir que soluções

boas não sejam descartadas logo de início. Desta forma mesmo ao utilizar alguma heurística para a geração da população inicial é primordial que parte da população seja gerada de forma aleatória.

Conforme Holland [27], a diversidade na população deve ser almejada independentemente se soluções são consideradas boas ou ruins.

3.2.3 Cálculo de aptidão ou avaliação *fitness*

O cálculo de aptidão é utilizado para avaliar/mensurar o nível de adaptabilidade de cada indivíduo de uma população ao problema, afim de tentar garantir que os melhores indivíduos tenham mais chances de prosseguir ou propagar suas características para a geração seguinte. A adaptabilidade contribui na convergência do algoritmo em busca da melhor solução para o problema.

Para tal propósito faz-se o uso de uma função de avaliação, chamada função *fitness*, a qual, em geral, utiliza-se do valor da função objetivo no ponto representado pelos cromossomos do indivíduo.

A função objetivo é considerada como parte da descrição do problema, pela mesma ser diretamente influenciada pela especificidade de cada problema em estudo. Esta é a única informação presente no processo evolutivo e pode ser vista como a qualidade do cromossomo perante o problema abordado.

O valor retornado pelo cálculo da aptidão é utilizado pelo processo de seleção, onde indivíduos mais aptos terão maiores chances de se reproduziram ou seguirem para a próxima geração.

3.2.4 Seleção dos indivíduos

A operação de seleção em um algoritmo genético tenta imitar o processo de seleção natural presente na natureza, onde se tem a sobrevivência do melhor. Nessa operação são selecionados os indivíduos da população atual que servem de base para a construção da próxima população, ou seja, os elementos que terão suas características genéticas transferida para seus descendentes.

Nesta etapa procura-se promover os indivíduos mais adaptados ao ambiente, afim de encontrar uma melhor solução para o problema, porém, sem a exclusão dos menos adaptados a fim de preservar sua diversidade.

Na seção seguinte é mostrado com mais detalhes como funciona o método da roleta, também conhecido como método de Monte Carlo, o qual será utilizado nesse trabalho.

Método de Monte Carlo

O método de Monte Carlo, proposto por Goldberg [23] como meio de seleção, é um método probabilístico baseado no mecanismo de amostragem estocástica.

Neste método cada indivíduo tem uma probabilidade de ser selecionado, a qual é proporcional a sua aptidão ao problema e dos outros elementos da população, esta calculada com o auxílio da função objetivo.

Ainda de acordo com o mesmo autor, a seguir são detalhados os passos deste método:

- Cálculo do *fitness* relativo: nele tem-se que a soma do *fitness* de todos os indivíduos é igual a 100. A partir disso é efetuado o cálculo da Equação 3.1 para maximização ou da inversa da Equação 3.1 no caso de minimização, para se encontrar o *fitness* relativo do indivíduo em questão. Onde, i e j pertencem ao intervalo $[1,n]$, e n sendo o número de indivíduos na população;

$$fitness_{relativo}(i) = \frac{100 * fitness_i}{\sum_{j=1}^{n} fitness_j} \tag{3.1}$$

- Distribuição dos indivíduos na roleta: após o cálculo do *fitness* relativo, os indivíduos são dispostos em um círculo, onde a fatia de cada um é proporcional ao seu *fitness relativo*. Com isso elementos mais aptos recebem uma fatia maior da roleta;

- Distribuição dos ponteiro: nesta etapa são distribuídos n ponteiros na roleta, os quais são equidistantes entre si, e n sendo o número de indivíduos na população;

- Realização do giro: ao final, é executado o giro dos ponteiros de forma aleatória e, os indivíduos que estiverem sob os ponteiros serão os elementos selecionados pelo método.

3.2.5 Cruzamento (crossover)

Na recombinação, ou cruzamento, ocorre a troca de fragmentos entre pares de indivíduos selecionados anteriormente afim de se criar os novos indivíduos da população seguinte.

Esta etapa é tida como o mecanismo de exploração primordial de um algoritmo genético [10]. Sendo considerado a principal ferramenta de busca dos AGs para explorar regiões desconhecidas no espaço de busca [15].

O cruzamento de diferentes indivíduos tem por objetivo unir boas características de soluções diferentes e conseguir uma solução ainda melhor. Estas características boas podem estar presentes tanto em indivíduos bons quanto em indivíduos não tão bons.

A recombinação ocorre somente se a taxa de cruzamento for atingida. De acordo com Santa Catarina e Bach [42], é interessante estar entre 60% e 80% e, segundo Lacerda [15] entre 60% e 90%.

Assim como a codificação é dependente do problema, o tipo de cruzamento empregado é dependente da codificação utilizada. A seguir são apresentados os principais métodos de cruzamento da codificação real.

Além destes, Herrera, Lozano e Verdegay [26], mencionam a existência dos seguintes operadores: *Flat crossover*, *Linear crossover*, *Extended line crossover*, *Linear BGA crossover*, *Fuzzy Connectives Based Crossover (FCB)*.

Dados dois cromossomos $C_1 = ({c_1}^1, \ldots, {c_L}^1)$ e $C_1 = ({c_1}^2, \ldots, {c_L}^2)$, onde L é o número de genes no cromossomo, pode-se tratar o cruzamento conforme as operações seguintes.

No cruzamento simples (*Simple crossover*) [54] [35], um ponto i é escolhido aleatoriamente, $i \in [1, 2, \ldots, L-1]$, e constroem-se os elementos filhos, onde o primeiro filho terá os genes do primeiro pai até a posição i, e os genes do segundo pai da posição $i+1$ em diante, já no segundo filho temos os genes do segundo pai até a posição i e do primeiro pai da posição $i+1$ em diante, como se segue: $H_1 = ({c_1}^1, \ldots, {c_i}^1, {c_{i+1}}^2, \ldots, {c_L}^2)$ e $H_2 = ({c_1}^2, \ldots, {c_i}^2, {c_{i+1}}^1, \ldots, {c_L}^1)$.

No cruzamento discreto (*Discrete crossover*) [37], o cromossomo filho é gerado a partir da escolha aleatória de genes, na distribuição uniforme formada com os genes dos pais: $H = (h_1, \ldots, h_i, \ldots, h_L)$, onde $h_i \in \left[{c_i}^1, {c_i}^2\right]$.

No cruzamento por média simples [26], o cromossomo filho é gerado a partir da aplicação da média simples sobre seus pais, $H = (h_1, \ldots, h_i, \ldots, h_L)$, onde h_i é igual ao resultado da aplicação da Equação 3.2.

$$h_i = \frac{({c_i}^1 + {c_i}^2)}{2} \tag{3.2}$$

No cruzamento por média geométrica [26], o cromossomo filho é gerado a partir da aplicação da média geométrica sobre seus pais, $H = (h_1, \ldots, h_i, \ldots, h_L)$, onde h_i é igual ao resultado da aplicação da Equação 3.3.

$$h_i = \sqrt{{c_i}^1 * {c_i}^2} \tag{3.3}$$

No cruzamento aritmético (*Arithmetical crossover*) [35], os filhos são gerados a partir da aplicação das Equações 3.4 e 3.5 sobre seus pais, onde λ é constante. Para cruzamento aritmético uniforme, ou variante dependendo do número de gerações iteradas, para o cruzamento aritmético não uniforme.

$${h_i}^1 = \lambda {c_i}^1 + (1 - \lambda){c_i}^2 \tag{3.4}$$

$${h_i}^2 = \lambda {c_i}^2 + (1 - \lambda){c_i}^1 \tag{3.5}$$

No cruzamento heurístico (*Qright's heuristic crossover*) [46], o filho é gerado dando uma maior prioridade aos genes do pai com maior aptidão. Dado que o pai com maior aptidão é C_i temos que: $H = (h_1, \ldots, h_i, \ldots, h_L)$, onde h_i é igual ao resultado da aplicação da Equação 3.6 e r sendo um valor randômico do intervalo $[0, 1]$.

$$h_i = r * ({c_i}^1 - {c_i}^2) + {c_i}^1 \tag{3.6}$$

No cruzamento $BLX - \alpha$, o filho $H = (h_1, \ldots, h_i, \ldots, h_L)$ é gerado a partir de valores aleatórios, e uniformes, pertencentes ao intervalo $[c_{min} - I * \alpha, c_{max} + I * \alpha]$, onde c_{max} é obtido pela Equação 3.7, c_{min} a partir da Equação 3.8 e I através da Equação 3.9 e α é um pequeno valor que estende os limites para a definição de H.

$$c_{max} = max({c_i}^1, {c_i}^2) \tag{3.7}$$

$$c_{min} = min({c_i}^1, {c_i}^2) \tag{3.8}$$

$$I = c_{max} - c_{min} \tag{3.9}$$

3.2.6 Mutação

A operação de mutação tem por objetivo restaurar material genético, diversidade, perdida nas repetidas seleções e cruzamentos, ou transformá-lo, de forma que regiões do espaço de busca desconhecidas possam ser alcançadas [15], diminuindo desta forma as chances de convergência para máximos locais (soluções localmente boas mas não globalmente) e, assegurando que a probabilidade de atingir qualquer ponto no espaço de busca nunca seja zero [27].

Neste contexto, a mutação é um vetor que favorece a diversidade, permitindo que a busca por melhores soluções continue, sendo considerado, desta forma, um operador de busca secundário [10].

Embora melhore a diversidade dos cromossomos da população, a mutação acaba destruindo informações contidas no cromossomo. Por esse motivo, a probabilidade de mutação, de acordo com Santa Catarina e Bach [42], é interessante estar ente 0.5% e 5%, uma vez que valores maiores que 5% tornam a busca aleatória. Já Lacerda [15], sugere que a taxa de mutação deve ficar entre 0.1% e 5%, o que de acordo com o mesmo é o suficiente para assegurar diversidade.

O processo de mutação de um indivíduo tem início através da seleção, de forma arbitrária, randômica, de um gene do mesmo. Em seguida este gene selecionado é submetido à uma avaliação probabilística (baseada na probabilidade de mutação), para verificar se o mesmo sofrerá alteração no seu estado.

Assim como a codificação é dependente do problema, o tipo de mutação empregado é dependente da codificação utilizada. A seguir são apresentados os principais métodos de mutação da codificação real. Além destes, Herrera, Lozano e Verdegay [26], ainda mencionam a existência dos operadores *Muhlenbein's mutation* e *Contiuous modal mutation*.

Dados dois cromossomos $C = (c_1, \ldots, c_i, \ldots, c_n)$ e $c_i \in [a_i, b_i]$, onde n é o número de genes e $[a_i, b_i]$ um intervalo definido pelos limites mínimo e máximo permitido, a seguir é mostrado seu comportamento frente aos diferentes operadores.

Na mutação randômica (*Random mutation*) [35], o novo gene é escolhido de forma randômica no domínio $[a_i, b_i]$, este constituído de uma distribuição uniforme $c_i = x \in [a_i, b_i]$.

Na mutação gaussiana, segundo Herrera, Lozano e Verdegay [26], o novo gene é gerado a partir da distribuição $c_i = N(c_i, \sigma^2)$, onde c_i é o valor do gene a ser mutado, e σ é um valor definido pelo projetista. Os autores mencionam ainda, a existência de pesquisas que indicam

que o valor de σ deve ser iniciado próximo a 0.5 e a medida que novas gerações são iteradas ele pode ser diminuído.

Com a mutação *creep* (*Real Number Creep*) [13], tenta-se explorar máximos locais na busca da solução. Nesta, o gene é mutado adicionando-se ou subtraindo-se um pequeno fator gerado aleatoriamente. O valor máximo para este fator é definido pelo projetista, a fim de proporcionar tal exploração. $c_i = c_i \pm r$, onde c_i é o gene original e, r é valor aleatório. Herrera, Lozano e Verdegay [26] mencionam a existência de variações deste método, a qual se baseia no valor máximo de deslocamento permitido, como: *Guaranteed-Big-Creep*, *Guaranteed-Little-Creep*, *Small Creep* e *Large Creep*.

A mutação discreta modal (*Discrete modal mutation*), apresentada por Voigt [52], é obtida com a aplicação da Equação 3.10, onde π é obtido com o cálculo da Equação 3.11, em que $B_m > 1$ é a base da mutação e, $rang_{min}$ é o limite inferior da faixa de mutação relativa.

$$\gamma = \sum_{k=0}^{\pi} \alpha_k \cdot B_m^k \tag{3.10}$$

$$\pi = \left[log \frac{rang_{min}}{log(B_m)} \right] \tag{3.11}$$

Herrera, Lozano e Verdegay [26], ainda mencionam a mutação não-uniforme, onde o novo gene é um número selecionado do intervalo $[a_i, b_i]$, não uniformemente distribuído, e onde os números apresentam probabilidades de ocorrência diferentes. E a mutação não-uniforme múltipla, que apresenta os mesmos aspectos da mutação não uniforme com a diferença de que a mutação é realizada em todos os genes do cromossomo.

3.2.7 Elitismo

O processo de elitismo tenta garantir que indivíduos bons não sejam perdidos durante os processos de cruzamento e mutação, promovendo, para a próxima geração, um certo percentual dos melhores indivíduos antes que estas operações sejam executadas, ou seja, estes indivíduos têm seu material genético completamente copiado para a próxima geração sem alteração, conservando desta forma suas características consideradas boas. Com isso o tempo de convergência do algoritmo é melhorado, por outro lado, pode provocar uma convergência para um ótimo local.

3.2.8 Parâmetros Genéticos

Por se tratarem de problemas não lineares, em sua grande maioria, os parâmetros genéticos não podem ser tratados como uma variável independente, tão pouco serem resolvidos independentemente. Existem interações, entre os parâmetros, que devem ser consideradas para maximizar ou minimizar a saída da função fitness. A esta iteração dá-se o nome de epistase (epistasis) [3].

Os principais parâmetros genéticos são:

- **Tamanho da população**: o tamanho da população influencia diretamente a qualidade e o tempo de obtenção da solução. Em populações pequenas pode-se ter problema com a cobertura limitada do espaço de busca, problema este que se tenta evitar com o uso de populações maiores. Nestas consegue-se evitar a convergência prematura para soluções locais, porém o esforço computacional pode ser demasiadamente grande [23]. Tanomaru [49], em seu estudo afirma que populações com 50 a 200 indivíduos são capazes de resolver a maioria dos problemas, exceto os mais complexos;

- **Número de gerações**: de acordo com Pinho [17], a importância do número de gerações está no fato de que não se pode afirmar com exatidão que uma solução é a melhor alcançável para poder terminar o processamento, com isso este é um fator comumente utilizado para tal tarefa. O mesmo autor afirma que tal parâmetro deve ser determinado experimentalmente, pois o mesmo depende do problema. Mitchell [36], diz que valores comumente utilizados estão na faixa de 50 à 500 gerações;

- **Taxa de Cruzamento**: a taxa de cruzamento tenta balancear a necessidade de se obter indivíduos melhores, a partir do cruzamento de indivíduos já bons, e o risco de perder um indivíduo muito apto, devido ao cruzamento. Não há consenso quanto a este valor. Para Tanomaru [49], deve ser maior que 0.7, Mitchell [36], menciona o valor de 0.6. O ponto comum em ambos os estudo é que o uso de taxas muito pequenas tendem a tornar o algoritmo lento;

- **Taxa de Mutação**: introduz a chance de se explorar o espaço de busca, evitando assim a convergência para soluções locais ou estagnações. Tal taxa não deve ser muito alta,

pois isso pode resultar em uma busca aleatória, além de poder provocar a destruição de soluções boas. Do estudo de Lacerda [15] é possível verificar que valores de 0.1% a 5% para a probabilidade de mutação são os mais adequados;

- **Intervalo da Geração**: é o percentual da população atual que será substituída na próxima geração. Assim como no cruzamento e mutação, taxas muito altas podem provocar a perda de soluções boas e muito pequenas o aumento no tempo de processamento.

Capítulo 4

Algoritmos genéticos paralelos

Neste capítulo são agrupados conceitos sobre programas paralelos e algoritmos genéticos. Seu objetivo é mostrar como estas duas áreas se interligam e como podem ser utilizadas na resolução de problemas de otimização e busca, acelarando a obtenção de boas soluções.

4.1 Introdução

Mühlenbein Heinz [25], define algoritmos genéticos como sendo uma busca randômica paralela com controle centralizado, onde a parte centralizada é responsável pela operação de seleção, a qual necessita do *fitness* relativo de todos os indivíduos, resultando assim em um algoritmo altamente sincronizado e difícil de se implementar eficientemente em computadores paralelos.

Geralmente programas que utilizam algoritmos genéticos são capazes de encontrar resultados de qualidade em tempos aceitáveis. Porém, ao aumentar-se o tamanho da entrada, o número de iterações, ou o espaço de busca do problema, eles tendem a requerer fatias de tempo cada vez maiores. Devido a esses fatores tem-se estudado técnicas para tornar o algoritmo genético mais rápido, e uma destas técnicas é a aplicação do paralelismo.

Ao se observar os componentes de um algoritmo genético vê-se que o mesmo é intrinsecamente paralelo, principalmente pelo fato de se trabalhar com um conjunto de soluções, e não com apenas uma, e pela evolução dos indivíduos serem independentes entre si [31].

Operações como as de avaliação, recombinação e mutação são exemplos de processos que podem ser paralelizados, sendo executados de forma simultânea em mais de um elemento da população. De acordo com Kohlmorgen [31], devido à essas características a obtenção de um

speedup (aumento de velocidade na da execução), significativo não é complicado.

A exceção ao paralelismo intrínseco de um algoritmo genético está na operação de seleção. Essa etapa do processo de evolução utiliza-se de informações de nível global da população, para determinar a qualidade relativa dos indivíduos [31].

Embora o processo de seleção se utilize de informações globais dos indivíduos de uma população, o mesmo pode ser paralelizado utilizando-se de seleção localizada, como no modelo de vizinhança, ou distribuída, como no modelo ilha.

Uma das soluções para o problema da seleção é utilizar seleção distribuída ou descentralizada [25]. Onde cada indivíduo realiza o processo de seleção, buscando por um parceiro em sua vizinhança, deste modo o indivíduo passa a ser um elemento atuante na busca, e acaba por melhorar seu *fitness*, durante seu tempo de vida, realizando buscas locais.

Outra solução, de acordo com Cantú-Paz [10], são as implementações de algoritmos genéticos paralelo mais populares, é a organização em múltiplas populações evoluindo separadamente e, com a troca de alguns indivíduos ocasionalmente, organização esta conhecida como *multi-deme*, granularidade grossa ou algoritmos genéticos distribuídos.

Um outro problema que o uso de paralelismo em algoritmos genéticos tenta resolver é a falta de diversidade. Segundo Heinz [25], um dos grandes problemas de algoritmos genéticos é a convergência prematura, e na tentativa de resolver esse problema os algoritmos genéticos tentam forçar a diversidade em sua população, claramente violando a natureza biológica.

Algoritmos genéticos paralelos tentam introduzir tal diversidade de um modo mais natural, utilizando-se de uma população com uma estrutura espacial, como na natureza. Em tal estrutura o *fitness* e o cruzamento são restritos a vizinhanças, chamadas de *demes*.

Esse nome foi introduzido em genética populacional, onde é sabido que populações com uma estrutura espacial tem uma maior variedade que uma população panmítica, população onde todos os elementos podem relacionar entre si.

Esta ideia, porém, não é uma verdade absoluta para todos os estudiosos da área. Para Wright [54], o melhor modo de evitar cair nos melhores locais, é dividir a população em varias sub-populações. Já para Fisher [18], tal teoria não é válida. Uma melhor discussão sobre o assunto pode ser encontrada em [25].

Visando esclarecer o funcionamento de algumas destas abordagens serão detalhados os

métodos mais comumente utilizados.

4.2 Modelos de algoritmos genéticos paralelos

A diferença principal entre as várias abordagens de paralelismo em algoritmos genéticos está no modo como a população é concebida e como é feita a seleção dos pais para a produção dos novos indivíduos [31], podendo apresentar-se de três formas: modelo de paralelização global, modelo de ilhas, modelo de vizinhança.

No modelo mestre escravo, ou global, o comportamento do algoritmo genético não é alterado, diferentemente do que acontece nos modelos de vizinhança e ilha.

Enquanto no modelo global a seleção utiliza-se de toda a população, nos outros dois é utilizado apenas uma parte da população, a subpopulação. Ainda, no mestre escravo, o cruzamento pode ser realizado entre quaisquer dois elementos, ao passo que nos outros dois métodos essa operação é restrita a uma parte da população, o conjunto de elementos que constituem a subpopulação.

No modelo ilha a seleção é restrita a subpopulações que são disjuntas, com exceção dos bons elementos que são trocados. A troca de elementos se dá a cada x gerações e para y ilhas vizinhas a um percentual w de sua população. Já no modelo de vizinhança, a seleção é feita a partir de vizinhanças altamente sobrepostas [31].

Diferentemente dos algoritmos tradicionais onde a paralelização preserva o comportamento funcional do algoritmo sequencial, as abordagens para paralelização de um algoritmo genético tradicional apresentam uma nova estrutura algorítmica, as quais são apresentadas a seguir.

4.2.1 Modelo Mestre-Escravo

No modelo de paralelismo Mestre-escravo, também conhecido como algoritmo genético paralelo global, o comportamento do algoritmo é exatamente o mesmo do modelo serial, diferindo apenas em tempo de execução.

A população apresenta uma distribuição global panmítica (onde todos os indivíduos tem a possibilidade de cruzamento entre si), tal como o modelo sequencial. A operação de seleção também faz o uso de todos os elementos da população. Nele o paralelismo é inserido na avaliação do *fitness* ou aplicação de um operador genético.

A população do modelo mestre-escravo fica no mestre, o qual também é responsável por delegar tarefas para os escravos. No modo mais usual o mestre envia uma porção da população para cada escravo, este realiza o cálculo do *fitness* destes indivíduos e retorna os valores para o mestre.

Sua utilização é interessante em problemas onde a avaliação de um indivíduo requer bastante poder de computação. Ainda este modelo tem a vantagem de não alterar o comportamento do algoritmo genético clássico, e as teorias deste podem ser facilmente empregadas naquele.

Neste contexto também é possível distribuir as tarefas de operações genéticas entre os escravos, mas de acordo com Cantú-Paz [10], por serem operações relativamente simples o tempo gasto com comunicação provavelmente seria maior do que o tempo ganho em performance.

Uma das justificativa para a escolha deste modelo de granularidade grossa (*coarse-grained*) é, por ele ser simples de se implementar com o uso de um cluster e softwares de domínio publico, pouco esforço é necessário para a conversão do algoritmo genético sequencial para esse modelo [50].

Ainda, implementação deste modelo em uma arquitetura de memória distribuída ou compartilhada não é problema. A diferença está apenas no modo como os processadores obtém os dados para trabalhar.

4.2.2 Modelo Ilha

O modelo de paralelismo ilha, também conhecido por multi-populacional, multi-deme, algoritmo genético distribuído, dentre outros, é constituído de várias sub-populações, estas consideradas a principal característica deste modelo de acordo com Cantú-Paz [10], que trocam indivíduos ocasionalmente através da operação de migração.

A denominação algoritmo genético distribuído é devido ao fato de o mesmo ser implementado, geralmente, em sistemas com memória distribuída, MIMD.

O tamanho das subpopulações, ou *demes*, é menor do que a população de um algoritmo genético serial. Isso faz com que a convergência seja mais rápida, podendo resultar em uma solução mais pobre [10].

Neste modelo a população é dividida em subpopulações e cada uma destas é atribuída à um processador. Neste é executada uma instância modificada do algoritmo genético serial, de modo

que se tenha várias populações sendo melhoradas de forma concorrente, e de tempos em tempos pode haver a possibilidade de troca de indivíduos entre populações vizinhas [31].

A principal diferença para o modelo clássico está na aplicação da operação de migração, onde indivíduos bons são trocados entre as subpopulações a medida da necessidade. Segundo Tanese [47], migração muito freqüente ou pouco frequente degradam a performance do algoritmo. Conforme o estudo de Cohoon [9], nas gerações entre migrações consegue-se pouca modificação na população, no entanto novas soluções são encontradas rapidamente após a troca de indivíduos.

A operação de migração é controlada através de dois parâmetros de migração [47]:

- **Intervalo de migração** - o número de gerações entre cada migração;
- **Taxa de migração** - o percentual de indivíduos selecionados para migração.

A codificação extra para se transformar um algoritmo genético sequencial em um paralelo utilizando este modelo está na criação de rotinas para a operação de migração.

A arquitetura necessária para o uso deste modelo é a de granularidade grossa, facilmente conseguida com o uso de cluster ou multicomputadores [31].

De acordo com Cantú-Paz [10], este modelo apresenta diversas variações de implementação. Na mais simples delas, *Davidor's ECO model*, são executadas várias instâncias do algoritmo genético em computadores diferentes e ao final é feito a coleta dos resultados, não fazendo, deste modo, o uso da operação de migração.

4.2.3 Modelo de vizinhança

No modelo de paralelismo de vizinhança, ou de granularidade fina, a população é subdividida em um número grande de pequenas populações, as quais podem ser compostas de apenas um indivíduo.

Cada população neste modelo é mantida em um processador diferente. E, os processadores que fazem parte do ambiente são conectados em *array*. Essa arrumação espacial proporciona o uso natural de vizinhanças locais para a seleção dos pais para a próxima geração [31].

As operações de seleção e cruzamento são restritas a pequenas vizinhanças, as quais são sobrepostas de modo a possibilitar uma interação entre todos os indivíduos. Ainda, neste mo-

delo todas as decisões são tomadas pelos indivíduos, fazendo com que ele seja completamente distribuído e sem controle central.

O pseudo algoritmo genético paralelo usando o modelo de vizinhança é apresentado a seguir, [25]:

1. Definir uma representação genética do problema;
2. Criar a população inicial e sua estrutura populacional;
3. Cada indivíduo melhora sua solução localmente;
4. Cada indivíduo seleciona um parceiro a partir de seus vizinhos;
5. Gera-se um filho através do cruzamento dos indivíduos;
6. O filho melhora sua solução localmente e substitui o pai se for melhor;
7. Repete-se do quarto passo em diante até um critério de término ser atingido.

É o modelo de paralelismo ideal para arquiteturas massivamente paralelas (máquinas que tem um grande número de processadores básicos), com granularidade fina. Onde tem-se apenas uma população estruturada de forma espacial, o que limita a interação entre indivíduos, de modo que, o cenário ideal é o uso de um indivíduo em cada unidade processadora.

4.2.4 Modelo hierárquico

Esse modelo combina o uso de subpopulações com o modelo mestre-escravo ou com um modelo de granularidade fina. E é denominado de hierárquico pois ele é constituído de uma única população subdividida, multi-deme.

De acordo com Cantú-Paz [10], este modelo oferece uma melhor performance do que o uso de qualquer dos modelos sozinho, reduzindo o seu tempo de execução.

Capítulo 5

Metodologia

Neste capítulo são apresentadas as técnicas utilizadas no desenvolvimento do trabalho e como foi feita a implementação.

Um dos objetivos do trabalho foi construir duas bibliotecas genéricas que possam ser facilmente reutilizadas. Essas bibliotecas tratam de um algoritmo genético sequencial e outros dois paralelos, onde o usuário das mesmas define alguns parâmetros e funções que são dependentes do problema a ser resolvido.

Para tal flexibilidade a codificação do cromossomo foi desenvolvida de modo a permitir diferentes modos de representação conhecidos, como: real, inteira, binária, dentre outras, e de modo à permitir que o projetista possa redefinir os métodos de cruzamento, mutação e criação de cromossomos aleatórios.

Além do desenvolvimento das bibliotecas foi feito um estudo comparativo da qualidade da solução e do desempenho temporal de ambas as abordagens para a resolução do problema do caixeiro viajante.

5.1 O algoritmo serial

O algoritmo genético serial implementado de forma genérica segue a forma funcional de um AG clássico apresentado. A principal diferença está no modo de representação do cromossomo, o qual não se utiliza da codificação binária.

O cromossomo foi definido como sendo um vetor de números reais, assim como na codificação real. Porém, isso não restringe o seu uso apenas para problemas que se utilizem de tal codificação, pelo contrário, permite que o mesmo seja utilizado para emular a representação de

outros tipos de codificação, como a inteira ou binária.

Além do cromossomo, cada indivíduo contém duas outras propriedades, que são seu *fitness* e o *fitness* relativo. Na Figura 5.1 é apresentada a estrutura de um elemento.

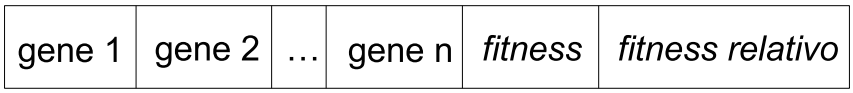

Figura 5.1: Estrutura do indivíduo implementado.

O valor *fitness* de um indivíduo é obtido através da função objetivo do algoritmo genético, a qual é dependente do problema, de modo que é a única função que deve, obrigatoriamente, ser definida pelo usuário da biblioteca. Já o valor do *fitness* relativo é encontrado calculando-se a proporção do *fitness* individual perante a população.

Além dos parâmetros normais encontrados em um algoritmo genético (como a taxa de cruzamento, mutação, elitismo, número de gerações e tamanho da população), a implementação em questão conta com um outro parâmetro, o limite dos genes, que se trata de uma estrutura que guarda os limites superiores e inferiores de cada gene de um cromossomo, de modo que nas operações às quais o mesmo é submetido, possam garantir que o espaço de busca não seja extrapolado.

Devido ao modo como foi concebida a estrutura do cromossomo, baseada em números reais, implementou-se também métodos para geração de cromossomos aleatórios, cruzamento e mutação baseados nos métodos para codificação real apresentados na seção pertinente.

A operação de geração de cromossomos aleatórios é necessária para a geração da população inicial. Ela faz o uso do parâmetro que guarda os limites dos genes, de forma a garantir que não sejas gerados cromossomos inválidos, fora do espaço de busca desejado. Esta operação pode ser definida pelo usuário da biblioteca se assim o quiser. Por exemplo, caso ele esteja se utilizando de um método heurístico para a geração dos elementos iniciais da população, ou esteja fazendo uso de uma emulação de codificação.

A operação de cruzamento implementada foi a média simples, a qual recebe dois cromossomos pais e constrói um cromossomo filho. Esse método de cruzamento foi escolhido devido a sua facilidade de implementação e por não gerar genes fora do espaço de busca, não necessitando assim de alguma rotina de readequação dos alelos.

Para a operação de mutação foi implementado o método de mutação gaussiana, onde o novo

valor para o gene é obtido a partir do sorteio de um valor aleatório pertencente ao intervalo definido por $|c_i - \sigma, c_i + \sigma|$, onde σ representa a variância definida pelo projetista e c_i o antigo valor do gene. Caso o novo valor do gene apresente-se fora dos limites estabelecidos, este é substituído pelo limite correspondente, através de uma operação de truncagem.

Caso se faça o uso da operação de mutação padrão (gaussiana), o algoritmo fará o uso de um outro parâmetro, a variância, o qual deverá ser definido pelo usuário da biblioteca. Este valor é utilizado na geração da distribuição onde é selecionado o novo valor do gene sendo mutado.

Assim como a operação de geração de cromossomo aleatório, as rotinas de cruzamento e mutação também podem ser definidas pelo usuário da biblioteca, principalmente no caso de este estar fazendo o uso de emulação de codificação.

Na operação de seleção pode utilizar-se o elitismo para promover alguns dos melhores indivíduos da população atual para a próxima geração sem sofrer processo de mutação/cruzamento. O número de elementos que serão promovidos é obtido com a aplicação da Equação 5.1, onde $taxa_{elitismo}$ é o tamanho relativo da elite e $tamanho_{pop}$ é o tamanho da população estudada, ambos definidos pelo projetista.

$$e = round(taxa_{elitismo} * tamanho_{pop}) \tag{5.1}$$

Na operação de seleção tem-se ainda uma peculiaridade; ao selecionar um indivíduo pai para o cruzamento já é feita a verificação da probabilidade do mesmo cruzar, de modo que tal taxa é verificada para cada elemento e não para cada par, e em caso negativo outro indivíduo é selecionado. Tal abordagem foi utilizada para fins de simplificação da operação de seleção.

Isso implica na verificação não ser feita pela operação de cruzamento, e tal ponto deve ser levado em consideração ao se definir um método de cruzamento próprio. Ainda, embora tal abordagem seja divergente da clássica, é considerada válida.

No processo de seleção dos indivíduos pais é utilizado o método da roleta, também conhecido como método de Monte Carlo, de modo que elementos da população mais aptos a se reproduzem tenham uma maior chance de serem escolhidos.

Na implementação do método de Monte Carlo, os indivíduos são distribuídos na roleta de acordo com seu *fitness* relativo, onde elementos com melhores valores recebem fatias maiores da roleta. Em seguida é realizado apenas um giro e espaçados *n* ponteiros sequencialmente ao

ponto onde a roleta parou, onde *n* é o número de indivíduos a serem escolhidos. Desta forma os elementos selecionados já são todos conhecidos.

Como resultado o algoritmo genético desenvolvido apresenta, através da saída padrão, os dados referentes ao indivíduo da população que obteve o melhor valor de *fitness*, ou seja, o elemento mais apto.

O fluxo de programa para a implementação serial pode ser observado a seguir.

1. Iniciar uma população;
2. Calcular o *fitness* dos indivíduos da população;
3. Selecionar indivíduos;
4. Verificar se indivíduo selecionada vai ser utilizado para cruzamento;
5. Recombinar os pares selecionados, afim de gerar os filhos;
6. Mutar os filhos;
7. Atualizar a população com os novos indivíduos;
8. Verificar se o ponto de parada foi encontrado, em caso negativo voltar para o passo 2;
9. Se ponto de parada atingido mostra o melhor indivíduo encontrado.

Na implementação do algoritmo genético sequencial o programa inicia gerando uma população de *n* indivíduos de forma aleatória. Esta população então é submetida iterativamente aos operadores genéticos de seleção, cruzamento e mutação, até que o número máximo de gerações seja atingido. Então, o algoritmo retorna o indivíduo que apresenta o melhor *fitness* para o problema especificado.

5.2 O algoritmo paralelo

Para a implementação do algoritmo genético escolheu-se duas abordagens paralelas. Na primeira tem-se o uso do tipo de paralelismo mais simples de se implementar, onde o mesmo algoritmo é executado em diversas instâncias concorrentes. Já para a segunda abordagem foi escolhido um modelo de arquitetura híbrido, onde tem-se a junção de dois tipos de paralelismo, o mestre-escravo com características da *multi-deme*.

O modelo híbrido desenvolvido será denominado de mestre-escravo no decorrer do trabalho por apresentar característica predominantemente de tal arquitetura.

5.2.1 Características do ambiente computacional disponível

Para o desenvolvimento das versões paralelas do algoritmo genético levou-se em conta o ambiente computacional dispovível para a realização dos testes.

O Laboratório de Computação de Alto Desempenho - LCAD - vinculado ao Grupo de Matemática Computacional e Processamento Paralelo - GMCPAR - pertencente ao Centro de Ciências Exatas e Tecnológicas - CCET - da Universidade Estadual do Oeste do Paraná - UNIOESTE - dispõe de um *cluster* do tipo *Beowulf*[1], para processamento de alto desempenho, no qual o objetivo é prover o aumento de processamento através da divisão de tarefas entre os diversos nós do *cluster* [40].

O *cluster*, nomeado krusty, é composto de 19 computadores, 18 destes exclusivos para processamento e 1 dedicado como interface de acesso, o *front-end*. Os equipamentos são equipados com processadores *Intel Pentium* 4 3.0GHz HT, memória cache de 1MB, RAM de 1GB em *dual channel* e disco rigido SATA de 80BG. A rede de interconexão utilizada para comunicação é do tipo *Gigabit-Ethernet*.

Como sistema operacional é utilizado o Fedora Core 8 da Red Hat Inc, e a biblioteca de computação paralela utilizada é a MPI em sua implementação MPICH, maiores informações sobre a configuração de *hardware* e *software* do krusty podem ser encontrados no trabalho de Ramos [40].

Ao executar um programa paralelo fazendo uso de MPI, aos processos nos nós é atribuído um identificador de processo, o qual é único e sequencial. Essa característica é observada para a definição do nodo que fará o papel de principal (número do processo igual á 0) e secundários (número do processo diferente de 0), em ambas as implementações realizadas.

5.2.2 Paralelismo simples

Esse modelo é caracterizado por inúmeros processos trabalhando sobre o mesmo problema de forma independente, evoluindo suas populações locais de forma concorrente, e ao fim do processamento, o melhor elemento é obtido entre os indivíduos produzidos por qualquer um dos processos.

[1] *Clusters* do tipo *Beowulf* são sistemas para computação paralela de alto desempenho. Foi desenvolvidos pela NASA em 1993 com o objetivo de utilizar-se somente de computadores pessoais e *software* livre [40]

A justificativa para a utilização deste modelo está no fato de ele ser a abordagem trivial, favorecendo a comparação com uma arquitetura mais recente, no caso a híbrida.

Essa metodologia também apresenta uma maior facilidade para a sua implementação, e as características do ambiente computacional necessário são atendidas pela configuração disponível para testes.

No algoritmo paralelo simples desenvolvido todos os processos possuem uma população local, incluindo o processo principal, gerada de forma aleatória por cada nó. Estes evoluem seus elementos, aplicando operadores de seleção, cruzamento e mutação, até que o ponto de parada seja atingido, que no caso é o número máximo de gerações, estabelecido inicialmente pelo projetista.

Numa comparação do tamanho da população do modelo sequencial com o paralelo simples, ambos têm o mesmo número de indivíduos, porém, este os subdivide em grupos menores para processamento.

Ao fim da computação todos os processos enviam o melhor indivíduo encontrado para o nó principal, o qual é encarregado de identificar o melhor elemento a partir dos cromossomos recebidos e exibí-lo via saída padrão.

A Figura 5.2 mostra o fluxo de execução da versão paralela simples implementada para um caso com 4 processadores.

Para esta versão paralela da biblioteca foram adicionadas rotinas para o envio e recebimento de elementos entre os processos.

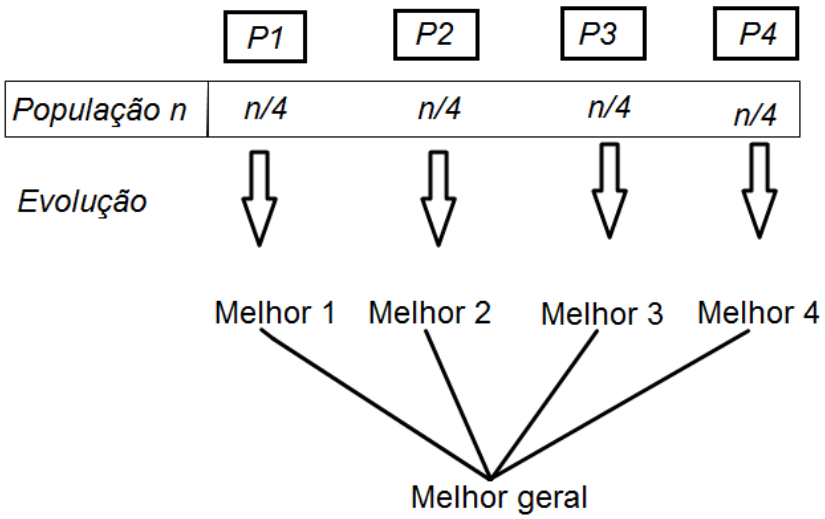

Figura 5.2: Fluxograma de execução da implementação do algoritmo paralelo simples.

A rotina de envio consiste em transferir os dados relativos ao cromossomo do indivíduo de melhor aptidão de um processo secundário para o processo principal. A rotina de recebimento preocupa-se em capturar os elementos recebidos, guardando-os em uma estrutura que é utilizada para achar o melhor entre todos os indivíduos. O processo principal não realiza o envio, nem o recebimento do melhor cromossomo gerado por si mesmo, guardando-o diretamente para posterior processamento.

A tarefa de envio e recebimento de dados entre os processos é facilitada com o uso das funções do MPI (`MPI_Send` e `MPI_Recv`), onde é necessário apenas definir o processo de origem (recebimento), destino (envio), o tipo de dados sendo transportado, seu tamanho e o endereço de memória onde o mesmo encontra-se ou será armazenado.

5.2.3 Paralelismo híbrido

Neste modelo tem-se a junção de outros modelos de arquitetura visando melhorar os resultados em termos de qualidade e consumo de tempo. O modelo híbrido desenvolvido faz uso dos modelos mestre-escravo com características dos modelos multi-deme.

A justificativa para a escolha do modelo de arquitetura híbrida com estas características está no fato de se ter a configuração do ambiente disponível para testes, o que torna propícia sua

implementação.

No algoritmo genético paralelo desenvolvido o mestre gera um população inicial e a divide entre seus escravos. Estes por sua vez evoluem sua subpopulação, aplicando os operadores de seleção, cruzamento e mutação, e a retornam para o mestre, o qual junta todas as subpopulações e redistribui de forma aleatória para os escravos.

A quantidade de vezes que será realizada a comunicação do mestre com os escravos, para distribuição e recolhimento dos indivíduos, é controlado pelo parâmetro genético, de número de iterações. Este é um vetor que propicia a troca de material genético entre subpopulações.

Neste trabalho o número de iterações é obtido dividindo-se o número de gerações por 50. Com isso 50 passa a ser a quantidade de vezes que cada nó escravo evolui uma subpopulação antes de a reenviar para o processo mestre.

Tal valor foi escolhido baseado no estudo de Cohoon [9], o qual diz que novas solução são encontradas rapidamente com a introdução de novos indivíduos na população, e ao manter-se o mesmo conjunto de elementos tende-se a diminuir esta rapidez. De modo que durante 50 gerações o material genético de uma subpopulação é trabalhada até uma certa estabilidade na diferença de qualidade entre gerações, e em seguida tenta-se fazer com que melhores soluções sejam encontradas mais rapidamente, misturando os indivíduos de diferentes subpopulações.

O final do processamento é dado quando o número de iterações desejado for atingido. A Figura 5.3 mostra o fluxo de execução da versão paralela mestre-escravo implementada para um caso com 4 processadores (o mestre e 3 escravos).

Para esta versão paralela da biblioteca foram adicionadas rotinas para o envio e recebimento de elementos entre os processos. E também foi implementado um módulo para realizar as atribuições do nodo mestre.

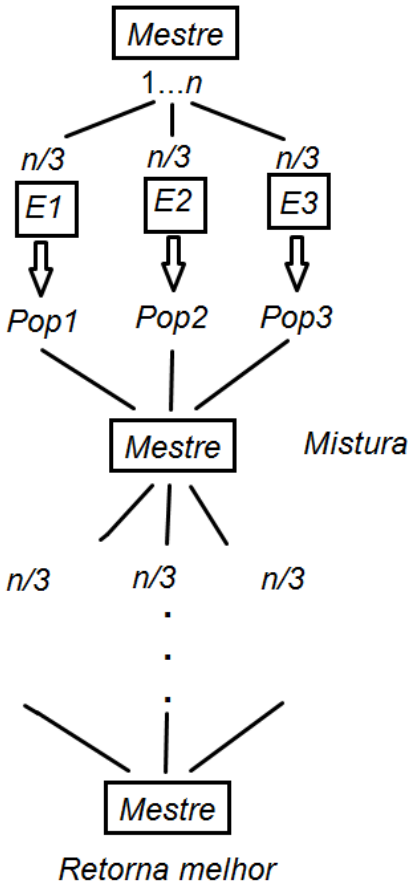

Figura 5.3: Fluxograma de execução da implementação do algoritmo paralelo mestre-escravo.

A rotina de envio e recebimento, consiste em transferir os dados relativos ao cromossomo de todos os indivíduos de uma subpopulação. Ela é executada tanto pelo processo mestre, ao realizar a distribuição dos elementos para os nós secundários e para o seu recebimento após as evoluções, ou pelos escravos, para receber os dados a serem processados e para retornar os resultados para o processo principal.

A tarefa de envio e recebimento de dados entre os processos é facilitada com o uso das funções do MPI (`MPI_Send` e `MPI_Recv`), onde é necessário apenas definir o processo de origem (recebimento), destino (envio), o tipo de dados sendo transportado, seu tamanho e o endereço de memória onde o mesmo se encontra ou será armazenado. A diferença para o modelo paralelo simples está no fato de se ter que realizar o envio e recebimento de mais de um indivíduo de cada vez, com isso as rotinas do MPI são chamadas diversas vezes para cada etapa de comunicação.

O processo mestre, com identificador 0, tem atribuições diferenciadas das dos processos escravos, não realizando evoluções diretamente. Ele é responsável por produzir os indivíduos

iniciais da população, distribuí-los, juntar os resultados e reenvia-los de forma aleatória para os escravos. Ao final de sua execução ele ainda seleciona o melhor elemento como sendo o resultado do programa, gerando sua saída via saída padrão.

5.3 Testes realizados

Nesta seção é apresentado o problema utilizado como estudo de caso, a teoria para a sua resolução através de algoritmos genéticos e como os testes foram realizados. Os resultados e análises são apresentados no capítulo seguinte.

5.3.1 Problema do caixeiro viajante

O problema do caixeiro viajante foi escolhido pois o mesmo faz parte da mesma família dos problemas de coloração de grafo (*graph drawing*), particionamento (*partitioning*) e agendamento (*scheduling*), e é tido como o pai de todos eles.

De acordo com Michalewicz [35] o problema do caixeito viajante tem sido aplicado em desenhos de placas de circuito, exames de raio-x com características especiais, fabricação de VLSI, dentre outros.

O problema do caixeiro viajante (*traveling salesman*), largamente conhecido como TSP, consiste em um vendedor que deve visitar todas as cidades de seu território exatamente uma vez e retornar para a cidade de início. É conhecido o custo para viajar entre todos os pares de cidades. O objetivo é encontrar uma rota em que o custo seja o menor possível.

A solução para o problema consiste na permutação das n cidades do espaço de busca, e qualquer permutação é uma solução válida, desde que o grafo gerado seja completo.

Euler é considerado o pai do problema, pois em 1759 definiu o problema dos cavaleiros (os cavalos do jogo de xadrez), onde um cavalo deveria visitar todas as casas do tabuleiro apenas uma vez, o qual é similar ao problema do caixeiro viajante. Por isso ele também é chamado de caminho Euleriano.

5.3.2 Modelagem do problema

A seguir é apresentada a modelagem realizada durante o desenvolvimento da solução para o problema do caixeiro viajante.

Representação do cromossomo

A representação utilizada para o cromossomo é a combinação das cidades, que consiste em uma sequencia de números. Essa codificação é a mais natural para o problema e utiliza-se de números inteiros [35].

A representação binária também pode ser utilizada, porém, faz-se necessário a codificação e decodificação dos valores para strings de bits, e seu uso não representa benefícios. Ainda, operações de mutação ou cruzamento poderiam resultar em cadeias binárias válidas, contudo, rotas inválidas.

De acordo com Michalewicz [35] existem três modos principais de se representar uma rota do TSP utilizando-se de um vetor de números inteiros, adjacências (*adjacency*), ordinal (*ordinal*), e caminho (*path representations*). O método focado neste trabalho é o último, por este utilizar-se de uma representação mais natural de compreensão, e pela facilidade de se implementar a operação de cruzamento.

Na representação por vetor de caminho, o vetor de números indica uma rota $R = \{\ldots, c_i, c_{i+1}, \ldots, c_n\}$, onde c_{i+1} representa a próxima cidade do trajeto partindo de c_i.

A Figura 5.4 mostra uma rota e sua devida codificação utilizando-se deste modelo. Nela verifica-se que a rota denotada por $R = \{3, 1, 2, 4\}$ equivale a tomar a cidade 3 como ponto de partida, seguir para 1, de 1 ir para 2, em seguida chegar em 4, e por fim retornar para o ponto 3.

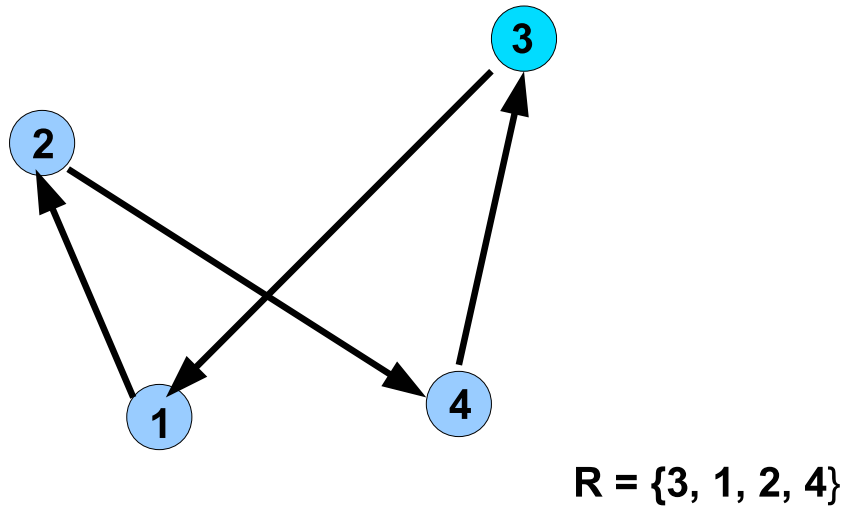

Figura 5.4: Exemplo de rota entre cidades e sua representação pelo método do caminho.

Função objetivo

A facilidade e naturalidade de representação, utilizando vetor de caminho, reflete no cálculo da função objetivo. Bastando apenas calcular a soma das distâncias entre as cidades no percurso produzido.

Para o cálculo da função objetivo utilizou-se da soma da distância Euclidiana entre as cidades, representada pela Equação 5.2, onde $n+1=1$.

$$fitness = \sum_{i=1}^{n+1} \sqrt{(x_{c_{i+1}} - x_{c_i})^2 + (y_{c_{i+1}} - y_{c_i})^2} \tag{5.2}$$

Operadores genéticos

Para a representação por permutação Michalewicz [35] recomenta o operador de mapeamento parcial (PMX) para a realização do cruzamento, já para a mutação ele diz que qualquer método que não gere uma rota inválida pode ser utilizado, visto que uma perturbação pode ser facilmente produzida devido ao modelo de representação.

Devido a esse fato, a operação de mutação consiste em se escolher duas posições na rota e fazer a troca simples na ordem das mesmas, como pode ser visto na Figura 5.5, onde as cidades 2 e 4 da rota foram escolhidas aleatoriamente para serem mutadas.

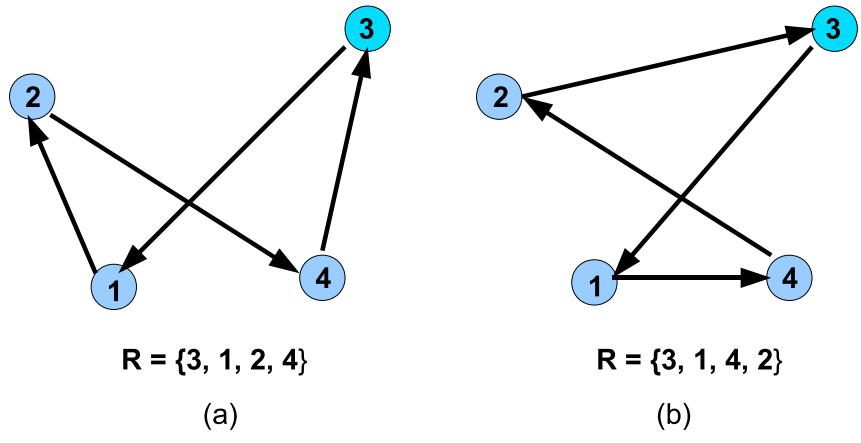

Figura 5.5: Exemplo de mutação de uma rota. (a) antes da mutação e (b) depois da mutação.

O operador de cruzamento utilizado, PMX, proposto por Goldberg e Lingle [22], consiste em criar um filho com uma subsequencia de um dos pais, e conservar a ordem e a posição do máximo de cidades que forem possíveis do outro pai.

Dados dois indivíduos pais $A = \{1,7,4,6,2,3,5\}$ e $B = \{6,4,5,7,2,3,1\}$, o filho é obtido seguindo-se os passos apresentados:

a) define-se duas posições de corte, de forma aleatória e diferentes entre si, para os pais $A = \{1,7,\|4,6,2,\|3,5\}$ e $B = \{6,4,\|5,7,2,\|3,1\}$, onde $\|$ representa o local de corte;

b) definir de forma aleatório que será o pai 1 e quem será o pai 2;

c) em seguida o conteúdo entre o corte de do pai 1 é copiado para o filho, $F = \{x,x,4,6,2,x,x\}$;

d) o próximo passo é gerar mapeamentos para os genes do pai 1 dentro da área de corte para genes do pai 2 na mesma posição, de modo que valores utilizados do pai 1 na etapa (c) sejam mapeados para valores relativos do pai 2.

$$\begin{array}{lccccccccc} A = \{ & 1 & 7\| & 4 & 6 & 2 & \|3 & 5 & \} \\ & & & \downarrow & \downarrow & \downarrow & & & \\ B = \{ & 6 & 4\| & 5 & 7 & 2 & \|3 & 1 & \} \end{array}$$

Gerando os seguintes valores para mapeamento:

$$\begin{array}{c} 4 \rightarrow 5 \\ 6 \rightarrow 7 \\ 2 \rightarrow 2 \end{array}$$

e) copia-se os genes do pai 2 que encontram-se fora da área de corte para o filho, se o valor já se encontrar no cromossomo filho é utilizado o valor correspondente obtido através do mapeamento realizado na etapa anterior, $F = \{7,5,4,6,2,3,1\}$;

Em seu livro Michalewicz [35], após fazer um estudo sobre os modos de representação e os operadores genéticos para o problema do caixeiro viajante, afirma que a busca por métodos e modelos mais eficientes ainda está em curso.

Considerações da implementação

Para a implementação da solução para o problema do caixeiro viajante foi utilizada a emulação do tipo de codificação, pois utilizou-se um vetor de números inteiros. Devido a este fato operações dependentes do modelo de codificação do problema (vetor de camimho) foram implementadas, como a função de mutação, cruzamento, geração de cromossomo aleatório, e a função objetivo, a qual já é particular ao problema por definição.

A implementação destes métodos e da emulação seguiu o modelo proposto por Michalewcz [35]. Essas funções são as mesmas para todas as versões das bibliotecas, serial, paralela mestre-escravo e paralela simples.

Para a geração de uma rota aleatória é construindo um percurso sequencial, em seguida são aplicadas perturbações semelhante àquelas da operação de mutação.

Para a operação de elitismo nos modelos paralelos a taxa definida pelo projetista é utilizada de forma local nos subprocessos e não globalmente. Ao se definir uma taxa de elitismo de, por exemplo, 2% para o programa todos os subprocessos aplicarão tal taxa em sua subpopulação.

5.3.3 Configuração dos testes

Nessa seção são apresentados o ambiente e o modo com que foram realizados os testes para a avaliação do desempenho do algoritmo sequencial e dos algoritmos paralelos implementados.

Durante o desenvolvimento deste trabalho o sistema estava configurado com 14 computadores (1 *front-end* e 13 nodos secundários), e a rede de interconexão disponível era do tipo *Fast-Ethernet*.

Para a execução dos testes paralelos utilizou-se 4 e 12 nodos do krusty, um dos 13 disponíveis foi deixado de reserva para eventuais problemas. Também foi utilizado a diretiva de execução do MPI `-nolocal`, que indica que o *front-end* não deve ser utilizado para rodar o programa na versão paralela. Já na versão sequencial utilizou-se apenas do *front-end* para realizar os testes.

Em todos os testes a população inicial utilizada pelos programas foi sempre a mesma, lida de um arquivo de entrada. Isso garante que todos as execuções tenham o mesmo estado inicial e sejam suscetíveis apenas às características do algoritmo genético.

Para o número de processos paralelos escolheu-se 4 e 12, tais valores foram escolhidos

por apresentarem um mesmo MDC (máximo divisor comum), permitindo que a população seja dividida de forma inteira em ambos os casos de teste.

No problema do TSP, para o cálculo da função *fitness*, é necessário saber a posição das cidades envolvidas no espaço, essas coordenas também são dados fixos e iguais para todas as execuções e versões do algoritmo.

Para a comparação adequada dos resultados, cada configuração foi executada 11 vezes, este valor é definido por Vieira [51] como sendo o número de repetições necessárias de um experimento para ele ser considerado confiável, quando apenas um fator estiver em estudo.

As configurações utilizadas focam na variação do número de indivíduos e no número de gerações, deixando alguns parâmetros genéticos inalterados nas diversas execuções de teste. Estes parâmetros e os valores utilizados podem ser observados na Tabela 5.1. Nela, *número de cidades* também é a dimensão do cromossomo trabalhado.

Tabela 5.1: Parâmetros genéticos fixos para todos os modelos em todas as execuções.

Parâmetro	**Valor adotado**
Número de cidades	1000
Percentual de elitismo	2%
Probabilidade de cruzamento	80%
Probabilidade de mutação	4%
Tipo de objetivo	0 (minimização)

Os parâmetros número de indivíduos e de gerações foram escolhidos como os parâmetros a serem modificados, por esses impactarem na cobertura do espaço de busca e evolução para uma solução melhor, respectivamente.

Os valores adotados, tanto para os parâmetros fixos quanto para o número de gerações, estão nos intervalos tidos como ideais por Mitchell [36], Tanomaru [49], Letamendia [32] e Lacerda [15].

Para os testes foram adotados três tamanhos de população 132, 528 e 1584 indivíduos. Tais valores foram escolhidos por terem os números de processos como múltiplos, 3 e 11 para o modelo mestre-escravo e 4 e 12 para o de paralelismo simples. E dois valores para número de gerações 200 e 500, estes escolhidos por serem múltiplos de 50, não causando imprecisão em relação ao número de iterações da implementação mestre-escravo.

As Tabelas 5.2 e 5.3 apresentam o tamanho das populações adotadas para 4 e 12 processos,

respectivamente, as quais são iguais. No entanto, o tamanho das subpopulações, o número de elementos sendo evoluídos nos nodos de cada vez, é dependente do tipo de modelo e da quantidade de processos. Pois, para a versão mestre escravo tem-se 3 ou 11 nós realizando trabalho de evolução, enquanto que na paralela simples tem-se 4 ou 12.

Na versão sequencial é adotado o tamanho da população global, que no caso pode ser 132, 528 ou 1853 indivíduos.

Tabela 5.2: Tamanho da polução global e das subpopulações para os modelos paralelos com 4 nodos de processamento.

Mestre Global	**Paralelo Simples Nodo**	**Paralelo Mestre-Escravo Nodo**
132	33	44
528	132	176
1584	396	528

Tabela 5.3: Tamanho da polução global e das subpopulações para os modelos paralelos com 12 nodos de processamento.

Mestre Global	**Paralelo Simples Nodo**	**Paralelo Mestre-Escravo Nodo**
132	11	12
528	44	48
1584	132	144

Com isso é possível visualizar o cenário de testes, onde o algoritmo sequencial foi executado com 132, 528 e 1584 indivíduos para 200 e 500 gerações. E as versões paralelas com 132, 528 e 1584 elementos para 200 e 500 gerações, divididos para 4 e 12 processos. O que totalizou 330 execuções de teste, 264 paralela e 66 sequencial.

Para realizar a medição do tempo gasto pelos processos para realizarem suas tarefas, nas versões paralelas foi utilizado a função do MPI `MPI_Wtime()`, e para o sequencial o comando `time` do *bash* do *linux*.

Para comparação da qualidade de cada configuração, será analisado o melhor indivíduo de cada execução, sendo o seu *fitness* armazenado para posterior estudo.

Capítulo 6

Resultados e discussões

Nesta seção são apresentados os resultados obtidos e discussões acerca deles. Foram avaliados o desempenho em relação ao tempo de execução e a qualidade das soluções obtidas, para a resolução do problema do caixeiro viajante utilizando-se os parâmetros definidos na Seção 5.3.3.

Para facilitar a apresentação dos dados foram calculadas as médias das 11 execução realizadas para cada configuração.

6.1 Tempo de execução

Para a avaliação do desempenho comparativo das versões paralelas com a serial, em relação ao tempo de execução, foram utilizadas as métricas *speedup* e eficiência, definidos na Seção 2.5.

As Tabelas 6.1 e 6.2 mostram as médias de tempo gasto para a execução com 132, 528 e 1584 indivíduos das versões sequencial, paralela simples com 4 e 12 nodos e paralela mestre-escravo com também 4 e 12 nodos, para 200 e 500 gerações respectivamente.

Tabela 6.1: Tempos de execução para 200 gerações.

População	**Serial**	**Simples 4 nós**	**Mestre 4 nós**	**Simples 12 nós**	**Mestre 12 nós**
132	17,51	4,83	6,97	2,43	3,20
528	70,45	17,98	26,85	7,00	9,68
1584	211,74	53,08	78,99	18,86	27,11

Tabela 6.2: Tempos de execução para 500 gerações.

População	Serial	Simples 4 nós	Mestre 4 nós	Simples 12 nós	Mestre 12 nós
132	42,69	11,18	16,14	4,47	6,46
528	170,82	42,8	63,65	15,57	21,48
1584	513,14	127,32	189,92	43,57	62,72

Através das Tabelas 6.1 e 6.2 é possível verificar que ambas versões paralelas apresentam um tempo de execução inferior ao obtido pelo modelo sequencial, em todas as configurações, como pode ser observado nos gráficos das Figuras 6.1 e 6.2. Isto se deve ao fato de que aquelas trabalham com subpopulações, enquanto a serial trabalha com toda a população em uma mesma máquina.

Essa característica faz com que o tempo desprendido em cada geração seja menor, são selecionados menos pais para cruzamento, realizadas menas operações de cruzamento e mutação.

Ainda é possível verificar que a versão paralela simples mostrou-se mais rápida em relação à paralela mestre-escravo, em todos os cenários. Isto se dá devido à comunicação extra empregada pelo modelo mestre-escravo, onde a cada 50 gerações é feito o envio de toda população para o processo mestre, e posteriormente recebida uma nova população, e pelo processamento adicional utilizado para misturar os indivíduos.

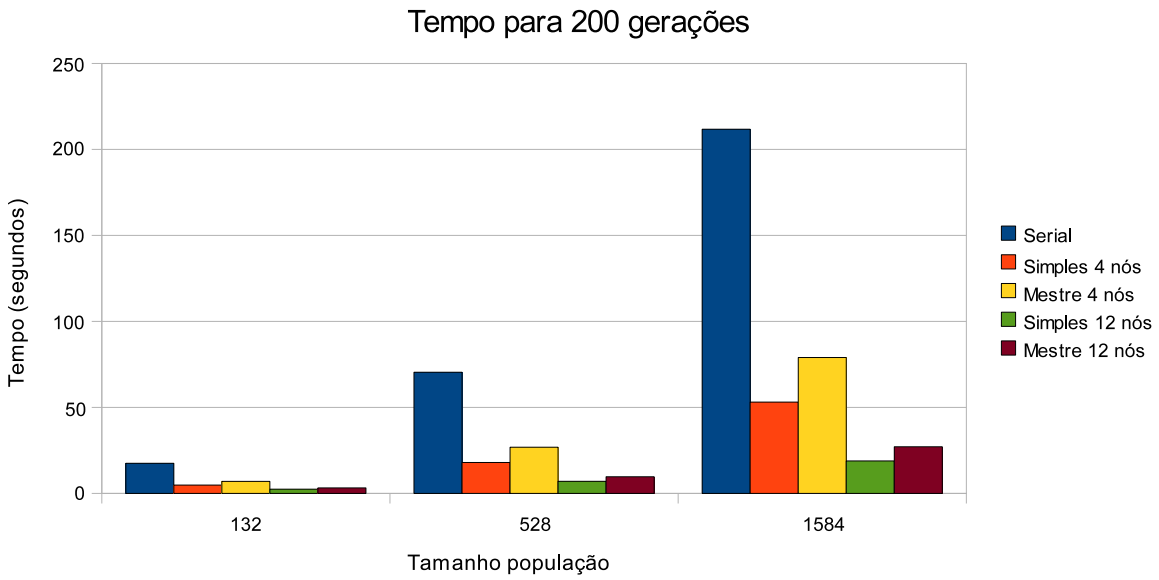

Figura 6.1: Tempos de execução para 200 gerações.

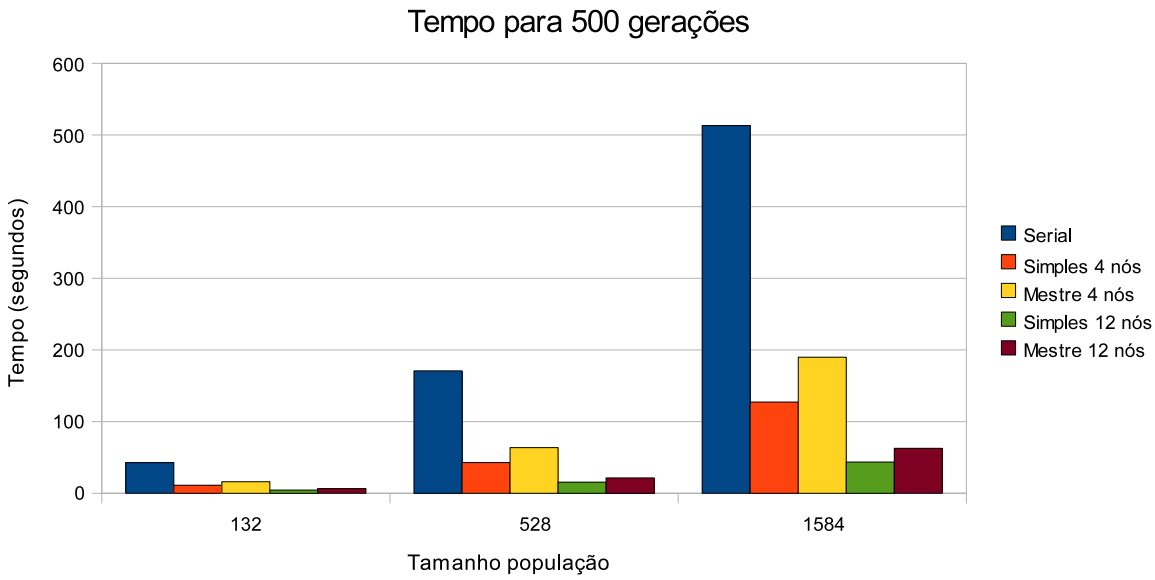

Figura 6.2: Tempos de execução para 500 gerações.

Nas Tabelas 6.3 e 6.4 são mostrados os *speedups* (quantas vezes mais rápido o trabalho foi completado) obtidos pelos modelos paralelos, em todas as configurações, em relação a versão sequencial.

Tabela 6.3: *Speedups* obtidos para 200 gerações.

População	**Simples 4 nós**	**Mestre 4 nós**	**Simples 12 nós**	**Mestre 12 nós**
132	3,63	2,51	7,21	5,47
528	3,92	2,62	10,06	7,28
1584	3,99	2,68	11,23	7,81

Tabela 6.4: *Speedups* obtidos para 500 gerações.

População	**Simples 4 nós**	**Mestre 4 nós**	**Simples 12 nós**	**Mestre 12 nós**
132	3,82	2,64	9,55	6,61
528	3,99	2,68	10,97	7,95
1584	4,03	2,70	11,78	8,18

Através das Tabelas 6.3 e 6.4 é possível verificar que aumentando-se o tamanho da população ou o número de gerações, a quantidade de vezes que o trabalho é realizado mais rápido (*speedup*), pelas versões paralelas, aumenta, como pode ser melhor observado nos gráficos das Figuras 6.3 e 6.4. Um dos motivos para tal resultado pode ser devido ao trabalho extra desprendido pelo algoritmo sequencial para a realização dos cálculos. Já nos modelos paralelos esse

fluxo extra de dados é dividido entre os processos resultando em um menor impacto.

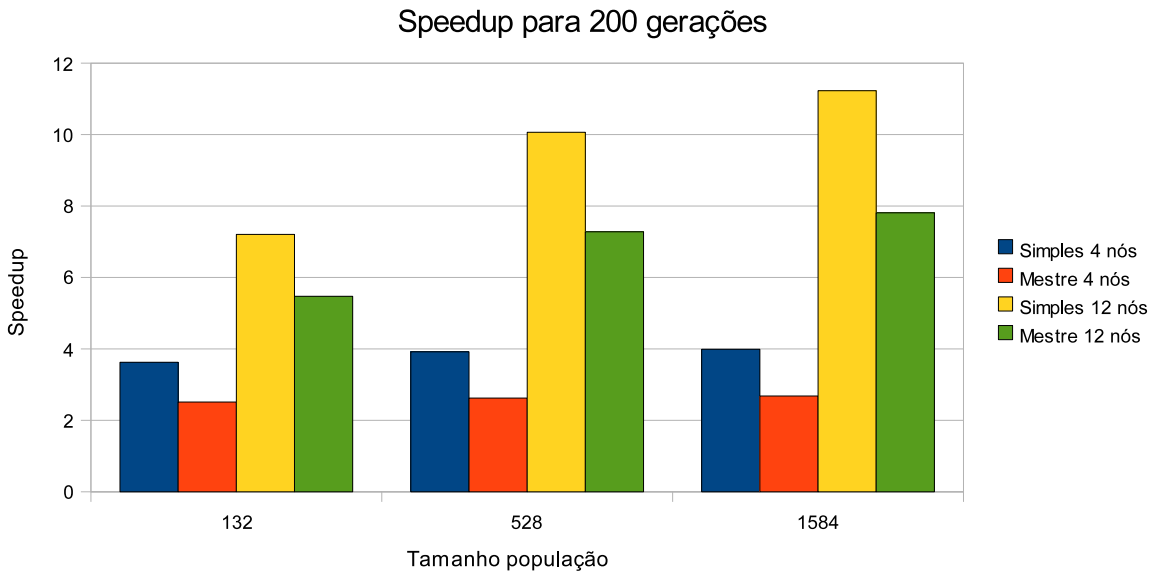

Figura 6.3: *Speedups* obtidos para a execução com 200 gerações.

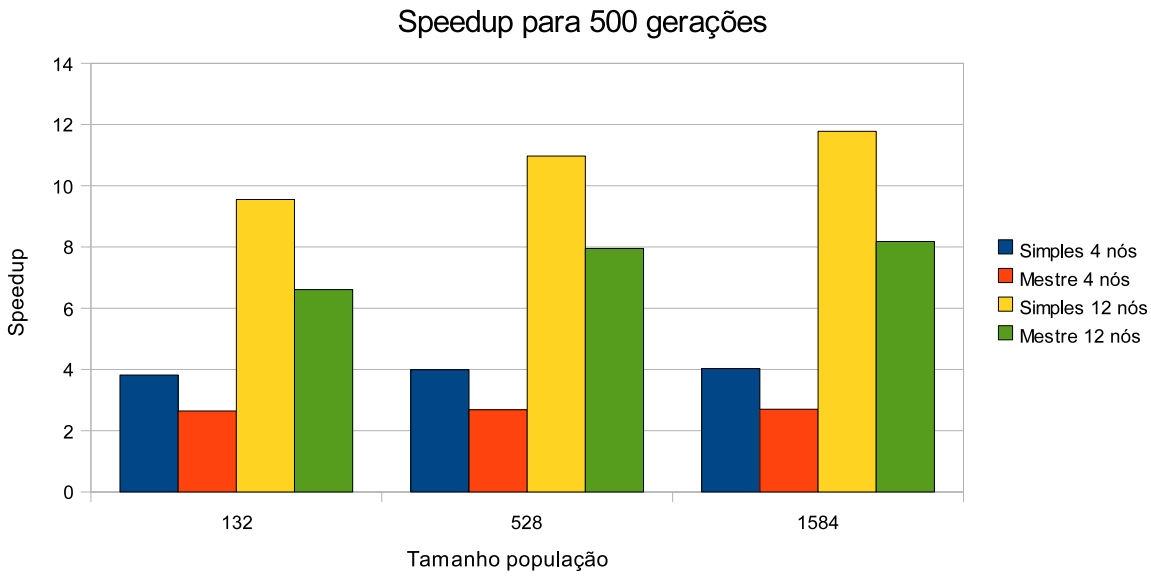

Figura 6.4: *Speedups* obtidos para a execução com 500 gerações.

Nas Tabelas 6.5 e 6.6 são apresentados os comparativos finais dos modelos implementados em relação a eficiência obtida. A medida de eficiência relaciona o *speedup* com o número de processadores utilizados [39], ou seja, mostra o quão válido é o emprego de mais nodos.

Tabela 6.5: Eficiências obtidas com 200 gerações.

População	Simples 4 nós	Mestre 4 nós	Simples 12 nós	Mestre 12 nós
132	0,91	0,63	0,74	0,46
528	0,98	0,66	0,84	0,61
1584	1,00	0,67	0,94	0,65

Tabela 6.6: Eficiências obtidas com 500 gerações.

População	Simples 4 nós	Mestre 4 nós	Simples 12 nós	Mestre 12 nós
132	0,95	0,66	0,80	0,55
528	1,00	0,67	0,91	0,66
1584	1,01	0,68	0,98	0,68

Através das Tabelas 6.5 e 6.6 é possível verifica que o emprego de mais processadores para a resolução do algoritmo genético foi eficiente, para todas as configurações testadas e principalmente para a versão paralela simples, como pode ser melhor observado nos gráficos das Figuras 6.5 e 6.6.

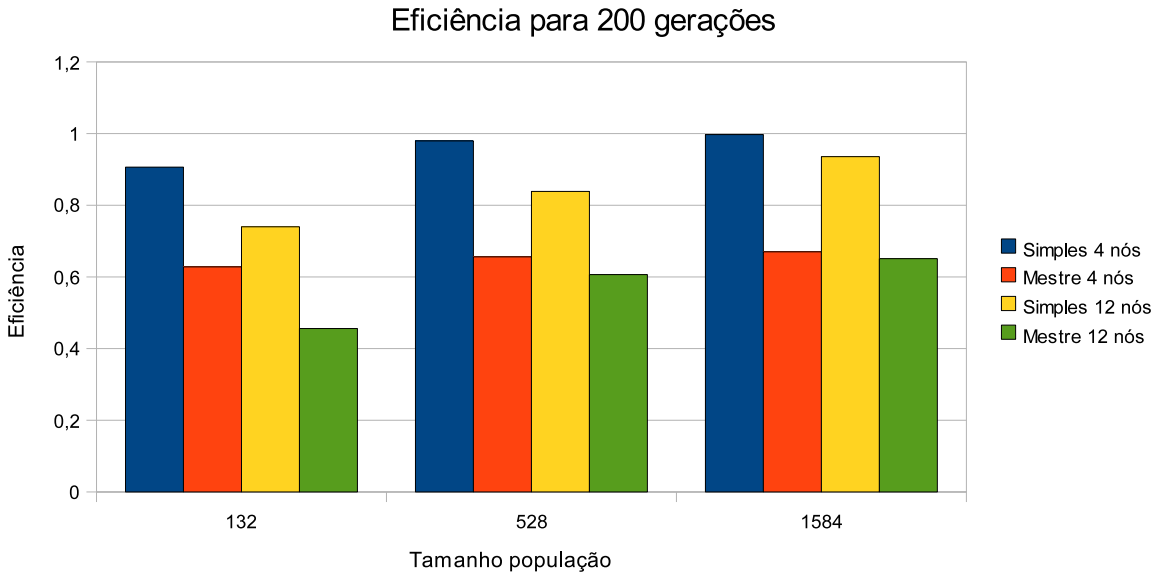

Figura 6.5: Eficiências dos modelos paralelos para 200 gerações.

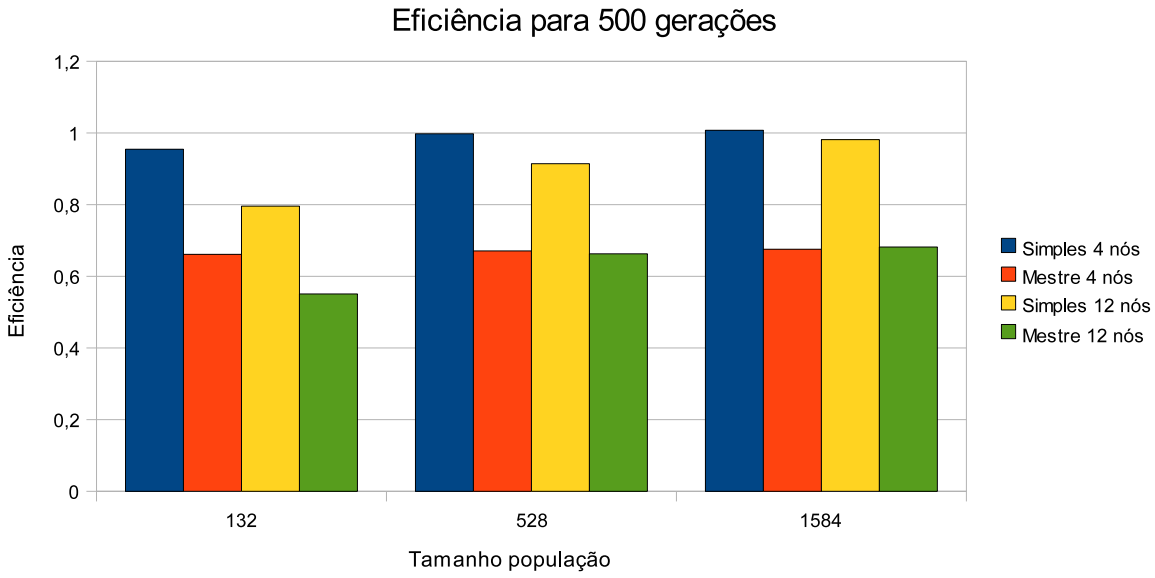

Figura 6.6: Eficiências dos modelos paralelos para 500 gerações.

O modelo paralelo simples foi também o que mais se beneficiou do aumento do número de gerações, mostrando melhora de performance mais significativa, se comparado com o modelo paralelo mestre-escravo.

Era esperado que a versão simples se saísse melhor na comparação, em relação ao tempo de execução, entre os modelos paralelos estudados. Nele tem-se um menor nível de comunicação, apenas ao final do processamento para transmitir a melhor solução encontrada, enquanto na mestre-escravo é feito o envio e recebimento de toda a população a cada 50 gerações. E por apresentar uma maior divisão do conjunto de dados, na versão mestre-escravo tem-se um nodo a menos realizando tarefa pesada, evolução de uma população, este fica dedicado ao processamento relativo ao mestre.

6.2 Qualidade da solução

Nesta seção são apresentados os resultados obtidos em relação a qualidade das soluções encontradas nas implementações desenvolvidas, aplicadas nos testes definidos. Para efeito de comparação é utilizado o *fitness* do melhor indivíduo da população, o resultado do algoritmo genético. Por ser um problema de minimização, quanto menor o valor do *fitness* melhor o indivíduo.

As Tabelas 6.7 e 6.8 mostram as médias das soluções obtidas com 132, 528 e 1584 indivíduos das versões sequencial, paralela simples com 4 e 12 nodos e paralela mestre-escravo com também 4 e 12 nodos, para 200 e 500 gerações respectivamente.

Tabela 6.7: *Fitness* dos indivíduos para 200 gerações.

População	**Serial**	**Simples 4 nós**	**Mestre 4 nós**	**Simples 12 nós**	**Mestre 12 nós**
132	480993,91	507919,97	493126,07	535892,52	516172,03
528	462305,14	477479,73	455528,52	499613,07	476013,05
1584	442958,04	459573,70	440981,24	473526,85	443876,31

Tabela 6.8: *Fitness* dos indivíduos para 500 gerações.

População	**Serial**	**Simples 4 nós**	**Mestre 4 nós**	**Simples 12 nós**	**Mestre 12 nós**
132	427102,10	471164,64	434240,58	508889,62	463422,67
528	390749,98	420013,94	379159,00	459795,03	395904,31
1584	363991,66	395845,50	353917,44	417892,63	354317,33

Através das Tabelas 6.7 e 6.8 é possível verificar que o modelo paralelo mestre-escravo apresenta uma qualidade média melhor do que as outras implementações e que no comparativo dos resultados para população com 132 indivíduos, a versão sequencial obteve a melhor solução para ambos os testes, isto pode ser melhor observado com o auxilio dos gráficos das Figuras 6.7 e 6.8.

Essa vantagem da versão sequencial pode ser devido ao elitismo. Em sua população, com 132 indivíduos, utilizando-se de uma taxa de 2%, tem-se 2 elelemtos promovidos para a próxima geração. Nos modelos paralelos o elitismo é avaliado localmente, fazendo com que, para uma população de 132 indivíduos, não se tenha elementos passados para a próxima geração sem sofrer a ação dos operadores genéticos de cruzamento e mutação, resultando em perda de matérial genético potencialmente bom.

Ao comparar a qualidade final obtida por cada configuração definida, verifica-se que a combinação 1584 indivíduos e paralelismo mestre-escravo com 4 processos produz os melhores resultados, tanto para 200 quanto para 500 gerações, obtendo o melhor desempenho global para 500.

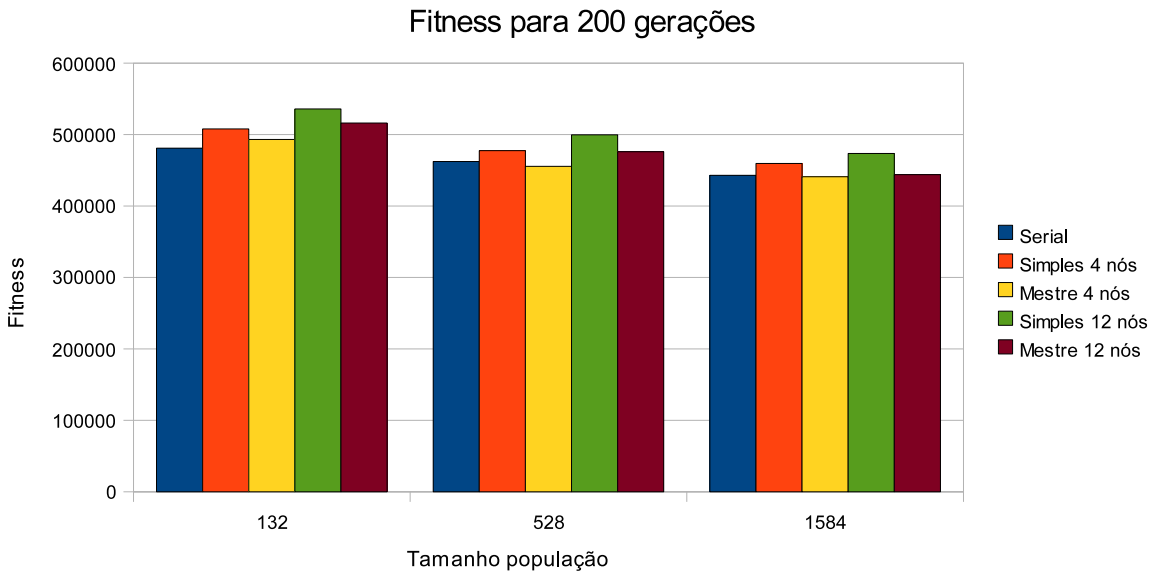

Figura 6.7: Qualidades dos indivíduos para 200 gerações.

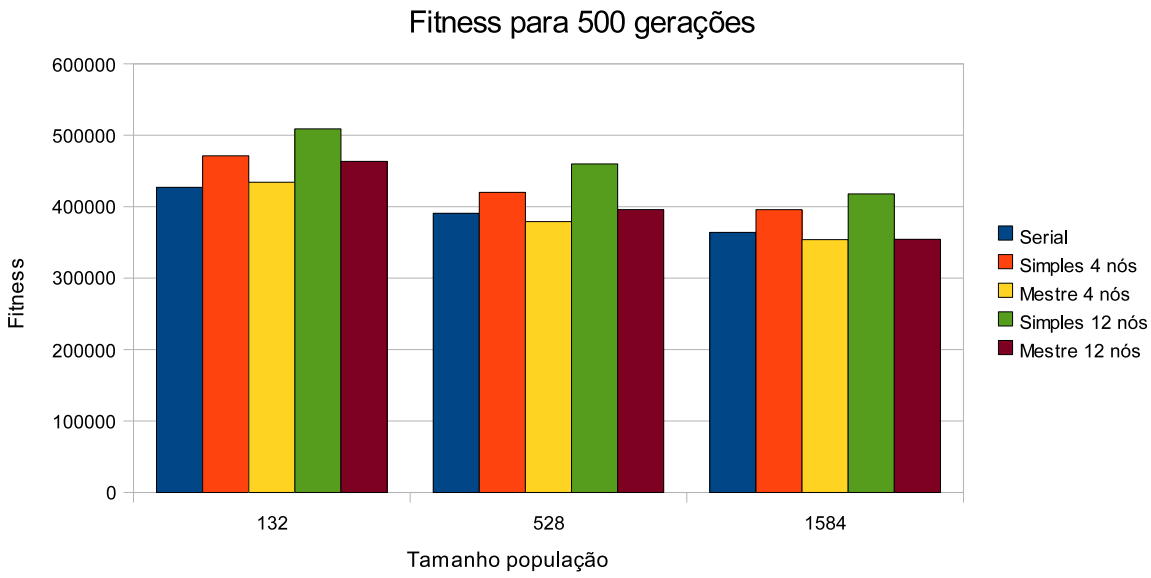

Figura 6.8: Qualidades dos indivíduos para 500 gerações.

As Tabelas 6.9 e 6.10 mostram a diferença de qualidade, de forma percentual, obtida nas versões paralelas com relação ao modelo sequencial. Valores negativo indicam uma solução pior e valores positivos um resultado melhor.

Tabela 6.9: Diferenças de qualidade para 200 gerações em relação a versão serial (%).

População	Simples 4 nós	Mestre 4 nós	Simples 12 nós	Mestre 12 nós
132	-5,60	-2,52	-11,41	-7,31
528	-3,28	1,47	-8,07	-2,97
1584	-3,75	0,45	-6,90	-0,21

Tabela 6.10: Diferenças de qualidade para 500 gerações em relação a versão serial (%).

População	Simples 4 nós	Mestre 4 nós	Simples 12 nós	Mestre 12 nós
132	-10,32	-1,67	-19,15	-8,50
528	-7,49	2,97	-17,67	-1,32
1584	-8,75	2,77	-14,81	2,66

Através das Tabelas 6.9 e 6.10 é possível verificar que a qualidade das soluções encontradas pelos modelos paralelos decai ao aumentar-se o número de processos concorrentes, e melhora à medida que o tamanho da população é aumentada. Reforçando assim a ideia de que quanto maior o número de indivíduos participando das evoluções melhores serão os resultados, pois o espaço de busca é melhor explorado.

No modelo mestre-escravo, conforme aumenta-se o número de indivíduos na população a qualidade da solução tende a equiparar-se com a obtida na versão serial, até um ponto onde a qualidade passa a ser melhor. Para ambas as execuções com 4 e 12 nodos. Tal fato é verificado timidamente no modelo paralelo simples, o qual obteve valores distantes dos obtidos pelas outras duas implementações.

O modelo mestre-escravo foi o que mais se beneficiou do aumento no número de gerações, entre os paralelos, mostrando melhora significativa da qualidade das soluções. Tal fato se deve à maior troca de elementos entre as subpopulações, fazendo com que se tenha uma maior interatividade entre os indivíduos da população global.

Para os casos estudados o uso do paralelismo com 4 processos mostrou-se melhor, quanto a qualidade das soluções encontradas e ao tempo de execução, porém, a medida que o tamanho da população é aumentado o ganho de tempo de qualidade tende a ser melhor para o uso de 12 processos.

Capítulo 7

Conclusão

No trabalho tinha-se como objetivo realizar a análise comparativa entre algoritmos genéticos sequenciais e paralelos, afim de verificar se o uso de paralelismo resultaria em melhoras no desempenho, em relação ao tempo de execução, e na qualidade das soluções encontradas.

Para cumprir este objetivo implementou-se uma biblioteca para a construção de AGs sequenciais e paralelos, um pequeno tutorial sobre como utilizá-las também foi escrito.

A versão sequencial do algoritmo genético seguiu a estrutura padrão de um AG e os modelos paralelos desenvolvidos foram o trivial, onde tem-se diversas instâncias distintas do mesmo AG sendo executado em conjuntos de dados diferentes, e híbrido, onde foram utilizadas características do modelo mestre-escravo - um processo coordenando os trabalhos e escravos realizando evoluções - e do modelos com múltiplas populações.

Na analise dos algoritmos observou-se que a versão paralela híbrida apresentou melhores resultados no quesito qualidade das soluções, em relação às versões sequencial e paralela simples, em diferentes configurações testadas (tamanho da população, número de gerações e número de nós).

Devido aos algoritmos genéticos não terem a garantia de encontrarem resultados exatos, mas geralmente aproximações, a versão paralela simples também representa uma boa opção de paralelismo a ser utilizada, pois suas soluções não foram ruins e foram obtidas em tempos reduzido.

Em relação ao modelo paralelo híbrido em especifico, melhores resultados, em relação ao tempo de execução, podem ser obtidos se for utilizado uma rede com capacidade melhor do que a utilizada, que foi Fast-Ethernet.

A conclusão em relação à qual é a melhor abordagem paralela fica relacionada à necessidade

do projetista, onde este deve definir o objetivo almejado, qualidade nas soluções, situação na qual o modelo paralelo híbrido deve ser o escolhido, ou menores tempo de execução, onde a versão paralela simples apresentou melhores resultados.

Entre os trabalhos futuros relevantes estão, fazer com que o mestre, do modelo híbrido, seja utilizado para a realização de processamento pesado, e não somente para redistribuição dos indivíduos para os escravos, isso pode afetar o tempo final de execução do programa e até fazer com que sejam obtidos valores próximos aos do modelo simples, onde tal característica já é utilizada.

Pode-se também implementar mecanismos que garantam diversidade nas novas populações nos modelos sequencial e paralelo simples, eliminando soluções similares.

Um novo estudo com outras estruturas de paralelismo também é relevante, permitindo que seja feito a comparação com as já implementadas. Assim como verificar a afirmação de Cohoon [9] que diz que a união de dois modelos paralelos gera melhores resultados do que qualquer um dos dois sozinho.

O estudo com outros problemas ou conjunto de dados que exijam maior poder computacional se mostra interessante, assim como a comparação com a solução ótima.

Um melhor tratamento do elitismo nos modelos paralelos também é um foco futuro, de modo a tentar garantir que o uso de pequenas populações não tenham sua qualidade final prejudicada.

No futuro pode-se refazer os testes utilizando-se de um número maior de nós, e tamanhos de populações diferentes, afim de confirmar os resultados obtidos.

Apêndice A

Tutorial de utilização das bibliotecas

O primeiro passo para se utilizar das bibliotecas construídas[1] é importar o arquivo de definição do algoritmo genético, essa arquivo recebe o nome de `AGSerial.h` para a versão serial, `AGParaleloSimples.h` para o modelo de paralelismo simples e `AGParaleloMestreEscravo.h` para a versão que utiliza-se da arquitetura mestre-escravo.

Em seguida deve-se definir a função objetivo, a qual será utilizada pelo algoritmo genético para calcular o *fitness* dos indivíduos. No algoritmo da Figura A.1, pode-se observar um exemplo de definição da função objetivo. Ainda, sua definição deve seguir um protótipo, o qual é apresentado a seguir.

```
/**Cela de Rosenbroch:  Z = -( 100 * (X^2 - Y)^2 + (1 - X)^2 ).
 onde os genes seriam X e Y e o elemento a ser maximizado seria o Z.**/

double objetivo(double* cromossomo, int tamanho){
    double x = cromossomo[0];
    double y = cromossomo[1];
    double fitness = - (
            100 * pow((pow(x, 2) - y), 2) + pow((1 - x), 2)
        );
    return fitness;
}
```

Figura A.1: Exemplo da implementação da função objetivo para uma equação de segundo grau, a Cela de Rosenbroch.

No protótipo da função objetivo o parâmetro cromossomo é um vetor de números reais, que contém os genes do indivíduo. E, o parâmetro tamanho refere-se à dimensão do vetor de genes, a dimensão do cromossomo. O retorno desta função deve ser o valor calculado, ou seja, o *fitness*

[1]Disponíveis em http://aggenerico.sourceforge.net/

do indivíduo.

Ao utilizar-se das bibliotecas é possível, também, definir funções de geração de cromossomos aleatórios, cruzamento e mutação, as quais devem seguir os protótipos apresentados na Tabela A.1 e detalhados a seguir.

Nome função	**Protótipo**
Objetivo	`double objetivo(const double* cromossomo, int tamanho);`
Cromossomo aleatório	`double* cromossomo_aleatorio(int tamanho, double** limite_genes);`
Cruzamento	`double* cruzamento(const double* mae, const double* pai, int tamanho, double** limite_genes);`
Mutação	`void mutacao(double* cromossomo, int tamanho, double** limite_genes, double taxa_mutacao);`

Tabela A.1: Tabela dos protótipos de função.

No protótipo da função de geração de cromossomo aleatório, o parâmetro tamanho refere-se ao tamanho do cromossomo a ser gerado, o número de genes. E, limite dos genes, é um vetor bidimensional onde cada posição guarda o limite inferior e superior que o gene do cromossomo na mesma posição pode assumir. O retorno deste método é um novo cromossomo com os valores aleatórios gerados.

No protótipo da função de cruzamento, os parâmetros mãe e pai representam os cromossomos que devem ser submetidos ao operador de forma a formar um filho. O tamanho refere-se a dimensão dos cromossomos participantes da operação, assim como o de seu filho. E, limite dos genes, é um vetor bidimensional onde cada posição guarda o limite inferior e superior que o gene do cromossomo na mesma posição pode assumir. O retorno deste método é um novo cromossomo com características de ambos os pais.

No protótipo da função de mutação, o parâmetro cromossomo representa os genes do indivíduo que sofrerão alteração genética. Tamanho refere-se a dimensão do cromossomo sendo tratado. O limite dos genes, é um vetor bidimensional onde cada posição guarda o limite inferior e superior que o gene do cromossomo na mesma posição pode assumir.

Ao se utilizar da redefinição destas funções descritas, deve-se tomar especial atenção ao limite dos genes, com o uso do parâmetro adequado, afim de não haver a geração de elementos na população que encontram-se fora do espaço de busca pesquisado.

No programa principal, a primeira chamada a ser realizada é a de inicialização do algoritmo genético. Esta função é responsável por realizar a pré-configuração do AG, que consiste em limpeza de variáveis e definição de funções.

A função a ser utilizada para tal objetivo tem o nome de `init_ag_serial()`, `init_ag_paralelo_simples()` ou `init_ag_paralelo_mestre_escravo()`, para a versão sequencial, paralela simples e paralela mestre-escravo, respectivamente.

Na função principal do programa o usuário da biblioteca deverá setar alguns parâmetro genéticos e funções auxiliares, eles podem ser obrigatório ou não. A Tabela A.2 relaciona os parâmetros disponíveis e o método de acesso, assim como o tipo de valor que a função recebe.

Parâmetro	**Função de acesso**	**Obrigatório**
Limite dos genes	`set_genes_limits(double**);`	Sim
Tamanho da população	`set_population_size(int);`	Sim
Tamanho do cromossomo	`set_chromosome_size(int);`	Sim
Número de gerações	`set_genarations(nger);`	Sim
Percentual de elitismo	`set_elitist_percentage (double);`	Não
Probabilidade de cruzamento	`set_crossover_probability(double);`	Sim
Probabilidade de mutação	`set_mutation_probability(double);`	Sim
Tipo de objetivo	`set_objective_type(int);`	Sim
Função objetivo	`set_objective_function(double(*)(double*, int));`	Sim
Função gerador cromossomo aleatório	`set_get_random_cromossomo_function(double*(*)(int, double**));`	Não
Função de cruzamento	`set_crossover_function(double*(*)(const double*, const double*, int, double**));`	Não
Função de mutação	`set_mutation_function(void(*)(double*, int, double**, double));`	Não

Tabela A.2: Tabela dos protótipos de função.

Ao setar as funções auxiliares deve-se respeitar os tipos de dados que as mesmas trabalham, como mostrado na Tabela A.2, já os limite aceitos para os parâmetros genéticos são apresentados na Tabela A.3.

Parâmetro	**Limites**
Tamanho da população	100 - 100000
Tamanho do cromossomo	2 - 10000
Número de gerações	50 - 100000
Percentual de elitismo	0% - 100%
Probabilidade de cruzamento	0% - 100%
Probabilidade de mutação	0% - 100%
Tipo de objetivo	0 (minimização) - 1 (maximização)

Tabela A.3: Limite dos parâmetros genéricos aceitos.

O passo seguinte é fazer a execução do algoritmo genético, o que pode ser obtido através da chamada da função `run_ag_serial()`, `run_ag_paralelo_mestre_escravo()` ou `run_ag_paralelo_simples()` para a versão sequencial, paralela mestre-escravo e paralela simples, respectivamente.

Ainda, antes de se finalizar a função principal do programa é boa prática fazer uma chamada para a função de liberação de recursos utilizados durante o processamento, o que pode ser obtido através de `dispose_ag_serial()`, `dispose_ag_paralelo_mestre_escravo()` ou `dispose_ag_paralelo_simples()`, para a versão sequencial, paralela mestre-escravo e paralela simples, respectivamente.

O algoritmo da Figura A.2 exemplifica uma função principal fazendo uso da biblioteca desenvolvida, para a execução da versão paralela mestre-escravo.

Ao fazer uso da versão paralela deve-se verificar se o ambiente é compatível com o utilizado neste trabalho, descrito na seção 5.2.1. Para a execução deve-se seguir o modo utilizado em algoritmos paralelos que fazem o uso da biblioteca MPI implementação MPICH, o qual pode ser encontrado em [20].

```
int main(int argc, char** argv){
    MPI_Init(&argc, &argv);

    init_ag_paralelo_masterslave();

    set_objective_function(objetivo); //A função objetivo do problema
    set_genes_limits(NULL); //O limite superior e inferior de cada gene
    set_chromosome_size(2); //Tamanhho do cromossomo
    set_population_size(100); //Tamanho da população
    set_genarations(1000); //Numero de gerações
    set_crossover_probability(80.0); //Probabilidade de cruzamento
    set_mutation_probability(4.0); //Probabilidade de mutação
    set_elitist_percentage(2.0); //Porcentual da população que vai ser selecionada por elitismo
    set_objective_type(0); //Objetivo - minimização

    run_ag_paralelo_masterslave();
    dispose_ag_paralelo_masterslave();

    MPI_Finalize();
}
```

Figura A.2: Exemplo função principal para a utilização da biblioteca.

Referências Bibliográficas

[1] ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1989.

[2] ANDREWS, G. R.; SHINEIDER, F. B. Concepts and notations for concurrent programming. **ACM Computing Survey**, Ithaca, NY, USA, p.3–43, 1983.

[3] AZEVEDO, R. et al. Sexual reproduction selects for robustness and negative epistasis in artificial gene networks. **Nature**, [S.l.], p.87–90, 2006.

[4] BäCK, T. **Evolutionary ALgorithms in Theory and Practice**. Oxford, England: Oxford University Press, 1996.

[5] BEGUELIN, A. **PVM: Parallel Virtual Machine. A user's Guide and Tutorial for Networked Parallel Computing**. Cambridge, MA, USA: The MIT Press, 1994.

[6] BERKENBROCK, G. R.; ROJAS, M. A. T. Projeto de caixa-s utilizando algoritmo genético paralelo. [S.l.].

[7] BITTENCOURT, G. **Inteligência Artificial: Ferramentas e Teorias**. Florianópolis, SC, Brasil: Editora da UFSC, 1998.

[8] BOOTH, S. et al. **Introduction to the T3D - A one day course**. Edinburgh Parallel Computing Centre - The University of Edinburgh, Edinburgh, Scotland.

[9] COHOON, J. P. et al. Punctuated equilibria: a parallel genetic algorithm. **Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application**, Hillsdale, NJ, USA, p.148–154, 1987.

[10] CANTú-PAZ, E. A survey of parallel genetic algorithms. **Reseaux et Systems Repartis**, [S.l.], p.141–171.

[11] CENAPAD-SP. **Introdução ao MPI**. Centro Nacional de Processamento de Alto Desempenho, São Paulo, SP, Brasil.

[12] DARWIN, C. **On the origin of species by means of natural selection**. London, England: John Murray, 1859.

[13] DAVIS, L. **Handbook of Genetic Algorithms**. New York, NY, USA: Van Nostrand Reinhold, 1991.

[14] DHAR, V.; STEIN, R. **Seven Methods for Transforming Corporative Data into Business Intelligence**. London, England: Prentice Hall, 1997.

[15] DE LACERDA, E. G. M. Introdução aos algoritmos genéticos. **Sistemas inteligentes: aplicações e recursos hídricos e ciências ambientais**, Porto Alegre, RS, Brasil, p.99–150, 1999.

[16] DONGARRA, J. et al. **Sourcebook of Parallel Computing**. San Francisco, CA, USA: Elsevier Science, 2003.

[17] DE PINHO, A. F.; MONTEVECHI, J. A. B.; MARINS, F. A. S. Análise da aplicação de projeto de experimentos nos parâmetros dos algoritmos genéticos. **S&G Revista Eletrônica**, Rio de Janeiro, RJ, Brasil, p.314–325, 2007.

[18] FISHER, R. A. **The Genetical Theory of Natural Selection**. New York, NY, USA: Dover, 1958.

[19] FLYNN, M. J. Some computer organization and their effectiveness. [S.l.], p.948–960, Setembro, 1972.

[20] GALANTE, G. **MPI - Message Passing Interface - Básico**. Tutorial. Universidade Estadual do Oeste do Paraná - Curso Bacharelado em Informática.

[21] GALANTE, G. **Métodos de Decomposição de Domínios para a Solução Paralela de Sistemas de Equações Lineares**. Cascavel, PR, Brasil: Universidade Estadual do Oeste do Paraná, Fevereiro, 2004. Monografia.

[22] GOLDBERG, D. E.; LINGLE, R. Alleles, loci, and the tsp. **Proceedings of the First International Conference on Genetic Algorithms Their Applications**, Hillsdale, NJ, USA, p.154–159, July, 1985.

[23] GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Massachusetts, USA: Addison-Wesley, 1989.

[24] HAUPT, R. L.; HAUPT, S. E. **Practical Genetic Algorithms**. New York, NY, USA: John Wiley & Sons, 1998.

[25] HEINZ, M. Evolution in time and space - the parallel genetic algorithm. **Foundations of Genetic Algorithms**, San Mateo, Calif, p.316–337, 1991.

[26] HERRERA, F.; LOZANO, M.; VERDEGAY, J. L. Trackling real-coded genetic algorithms: Operators and tools for behavioural analysis. **Artificial Intelligence Review**, [S.l.], 1996.

[27] HOLLAND, J. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI, USA: University of Michigan, 1975.

[28] HTTP://WWW.PORTALSAOFRANCISCO.COM.BR. **Teorias da Evolução**. Disponível em <http://www.portalsaofrancisco.com.br/alfa/evolucao-dos-seres-vivos/teorias-da-evolucao-2.php>. Acesso em: 21 de Novembro de 2009.

[29] KIRNER, C. Sistemas operacionais para ambientes paralelos. **Anais da VIII Jornada de Atualização em Informática**, Porto Alegre, RS, Brasil, p.3–43, 1989.

[30] KIRNER, C. Arquitetura de sistemas avançados de computação. **Anais da Jornada EPUSP/IEEE em Sistemas de Computação de Alto Desempenho**, São Paulo, SP, Brasil, p.307–353, 1991.

[31] KOHLMORGEN, U.; SCHMECK, H.; HAASE, K. Experiences with fine-grained parallel genetic algorithms. [S.l.], 1996.

[32] LETAMENDIA, L. N. Fitting the control parameters of a genetic algorithm - an application to technical trading systems design. **european Journal of Operational Reserch**, [S.l.], p.847–868, Junho, 2007.

[33] LUCASIUS, C. B.; KATEMAN, G. Application of genetic algorithms in chemometrics. **3rd international Conference on Genetic Algorithms**, San Francisco, CA, USA, p.170–176, 1991.

[34] MCBRYAN, O. A. An overview of message passing environments. **Parallel Computing**, Amsterdam, The Netherlands, p.417–444, 1994.

[35] MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. New York, NY, USA: Springer-Verlag New York, Inc., 1996.

[36] MITCHELL, M. **An Introductin to Genetic Algorithms**. London, England: The MIT Press, 1999.

[37] MUHLENBEIN., H.; SCHLIERKMP-VOOSEN, D. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. **Evol. Comput.**, Cambridge, MA, USA, p.25–49, 1993.

[38] NAVAUX, P. O. A. Introdução ao processamento paralelo. **RBC - Revista Brasileira de Computação**, Rio de Janeiro, RJ, Brasil, p.31–43, Outubro, 1989.

[39] QUINN, M. J. **Designing Efficient Algorithms for Parallel Computers**. New York, NY, USA: McGraw-Hill Inc., 1986.

[40] RAMOS, F. G. **Desenvolvimento de uma Ferramenta Web para Gerenciamento de Clusters**. Cascavel, PR, Brasil: Universidade Estadual do Oeste do Paraná, Dezembro, 2008. Monografia.

[41] SANTANA, R. H. C. et al. **Computação Paralela**. Universidade de São Paulo - Departamento de Ciências de Computação e Estatística, São Carlos, SP, Brasil, Setembro, 1997.

[42] SANTA CATARINA, A.; BACH, S. L. Estudo do efeito dos parâmetros genéticos sobre a solução otimizada e sobre o tempo de convergência em algoritmos genéticos com codificação binária e real. **Acta Scientiarum: Technology**, Maringá, PR, Brasil, p.147–152, 2003.

[43] SANTA CATARINA, A. **Aplicação de Algoritmos Genéticos em Sistemas de Informação Geográficas**. São José dos Campos, SP, Brasil: INPE, 2004. Monografia.

[44] SILVA, A. J. M. **Implementação de um algoritmo genético utilizando o modelo de ilhas**. Rio de Janeiro, RJ, Brasil: Universidade Federal do Rio de Janeiro. Disertação.

[45] SNOW, C. R. **Concurrent programming**. New York, NY, USA: Cambridge University Press, 1990.

[46] SCHLIERKAMP-VOOSEN, D. Strategy adaptation by competition. **Proceedings Second European Congress on Intelligent Techniques and Soft Computing**, Hillsdale, NJ, USA, p.1270–1274, 1994.

[47] TANESE, R. Parallel genetic algorithm for a hypercube. **Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application**, Hillsdale, NJ, USA, p.177–183, 1987.

[48] TANENBAUM, A. S. **Distributed Operationg Systems**. New York, NY, USA: Prentice Hall, 1995.

[49] TANOMARU, J. Motivação, fundamentos e aplicações de algoritmos genéticos. **II Congresso Brasileiro de Redes Neurais**, Curitiba, PR, Brasil, p.373–403, 1995.

[50] TONGCHIM, S. Coarse-grained parallel genetic algorithm for solving the timetable problem. Bangkok, Thailand.

[51] VIEIRA, S. **Estatística Experimental**. São Paulo, SP, Brasil: Atlas, 1999.

[52] VOIGT, H. M. A multivalued evolutionary algorithm. **Techical Report tr 93-022**, Berkeley, CA, USA, 1993.

[53] WALKER, D. W. The design of a standard message passing interface for distributed memory concurrent computers. **Parallel Computing**, Amsterdam, The Netherlands, p.657–673, 1994.

[54] WRIGHT, A. Genetic algorithms for real parameter optimization. San Mateo, CA, USA, p.205–218, 1991.

[55] ZALUSKA, E. J. Research lines in distributed computing systems and concurrent computation. **Anais do Workshop em Programação Concorrente, Sistemas Distribuídos e Engenharia de Software**, [S.l.], p.132–155, 1991.