UNIVERSIDADE FEDERAL DO PARANÁ

ERIK ALEXANDRE PUCCI

ALGORITMOS GENÉTICOS APLICADOS AO PROBLEMA DE *JOB SHOP SCHEDULING*: UM ESTUDO EMPÍRICO

CURITIBA

2012

ERIK ALEXANDRE PUCCI

ALGORITMOS GENÉTICOS APLICADOS AO PROBLEMA DE *JOB SHOP SCHEDULING*: UM ESTUDO EMPÍRICO

Monografia de Trabalho de Graduação apresentada ao Programa de Bacharelado em Ciência da Computação, Departamento de Informática, Setor de Ciências Exatas, Universidade Federal Do Paraná.

Orientador: Prof. Dr. Eduardo Jaques Spinosa

CURITIBA

2012

AGRADECIMENTOS

Agradeço inicialmente aos meus pais pela educação e, incluindo minha irmã, pelo contínuo apoio durante toda a minha vida.

Agradeço à minha namorada e aos meus amigos pelo apoio e compreensão durante todo o curso, especialmente no período de elaboração deste trabalho.

Agradeço ao meu orientador pela oportunidade e conhecimento.

Vocês fizeram toda a diferença para me ajudar a superar esta etapa da minha vida. Sou-lhes muito grato.

RESUMO

Problemas de otimização pertencem à classe $\mathcal{NP}$-difícil e são, destarte, de difícil resolução por métodos determinísticos ou exatos. Para alcançar soluções satisfatórias usando menos recursos computacionais, heurísticas como os Algoritmos Genéticos podem ser usadas. Este trabalho faz uma análise dos Algoritmos Genéticos e seus modelos paralelos aplicados ao problema de *Job Shop Scheduling*, experimentando certos parâmetros e operações de forma a otimizar a qualidade das soluções encontradas.

Palavras-chave: Algoritmos Genéticos, Algoritmos Genéticos Paralelos, *Job Shop Scheduling*.

ABSTRACT

Optimization problems belong to the $\mathcal{NP}$-hard class and are, therefore, of difficult resolution by deterministic or exact methods. To find satisfactory solutions using less computational resources, heuristics like Genetic Algorithms can be used. This work analyzes Genetic Algorithms and their parallel models applied to the Job Shop Scheduling problem, experimenting certain parameters and operations with the goal of optimizing the quality of the solutions found.

Keywords: Genetic Algorithms, Parallel Genetic Algorithms, Job Shop Scheduling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3DHP-Side-chain	3-Dimensional Hydrophobic-Polar Side-chain
AHHPGA	Asynchronous Heterogeneous HPGA
API	Application Programming Interface
ASPARAGOS	Asynchronous PGA
CGA	Cellular GA
CGPGA	Coarse-Grained PGA
CPU	Central Processing Unit
CS	Conflict Set
DGA	Distributed GA
EA	Evolutionary Algorithm
FGPGA	Fine-Grained PGA
G&T	Giffler & Thompson
GA	Genetic Algorithm
GAUL	GA Utility Library
GE-HPGA	Grid-Enabled HPGA framework
GPGA	Global PGA
GPL	General Public License
GPMX	Generalized Partially Mapped Crossover
GPU	Graphics Processing Unit
GTF	Giffler-Thompson Focused
HFC	Hierarchical Fair Competition
HFC-PGA	HFC-based PGA
HPGA	Hierarchical PGA
JSS	Job Shop Scheduling
JSSP	JSS Problem
MIMD	Multiple Instruction, Multiple Data
MPI	Message Passing Interface
MSXF-GA	Multi-Step Crossover Fusion GA
NP	Non-deterministic Polynomial-time
OR	Operations Research
PFP	Protein Folding Prediction problem
PGA	Parallel GA
PPX	Precedence Preservative Crossover
PPXm	PPX modificado
RAM	Random Access Memory
RAP	Redundancy Allocation Problem
RNA	Ribonucleic Acid
RR	Round-Robin
SGA	Sequential or Standard GA
SUS	Stochastic Universal Sampling
VLSI	Very-Large-Scale Integration

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Eficiência no custo e tempo de produção é um importante fator para os sistemas industriais, sendo que o escalonamento da produção é um fator fundamental para promover essa eficiência [1]. Um dos principais e mais complexos problemas de escalonamento estudados nos últimos anos é o *Job Shop Scheduling*, notadamente pertencente à classe $\mathcal{NP}$-difícil e considerado um dos seus piores membros [2, 3].

Dada a complexidade desse problema, heurísticas e métodos não exaustivos são extremamente úteis para encontrar as melhores soluções possíveis com o menor custo computacional [4]. Assim, este trabalho tem como objetivo realizar um estudo empírico de operadores, parâmetros e um modelo paralelo para a resolução de problemas de *Job Shop Scheduling* através do uso de Algoritmos Genéticos.

Para isso, após a definição de conceitos essenciais, serão analisados quais os parâmetros e operadores que possibilitam obter soluções de maior qualidade ao problema de escalonamento aqui tratado, entre cruzamentos, mutações, quantidade de gerações, elitismo e um modelo paralelo. Serão feitas comparações e avaliações focadas em alguns dos aspectos fundamentais de Algoritmos Genéticos.

Esta monografia está organizada em 7 capítulos. O Capítulo 2 define alguns conceitos sobre Algoritmos Genéticos, entre os quais uma nomenclatura da área, operadores, parâmetros e aplicações. O Capítulo 3 define e analisa modelos de Algoritmos Genéticos Paralelos, discutindo as diferenças entre a versão sequencial e os modelos paralelos, estudos teóricos, parâmetros e aplicações. O Capítulo 4 define o problema de *Job Shop Scheduling* e qual a utilidade e relação dos Algoritmos Genéticos com esse problema computacional. O Capítulo 5 apresenta em detalhes todos os operadores implementados (geração da população inicial, cruzamento, mutação, migração) e experimentos realizados. O Capítulo 6 apresenta os resultados obtidos dos experimentos descritos no capítulo anterior e e uma análise sobre as valorações encontradas. Por fim, o Capítulo 7 apresenta as considerações finais do estudo proposto.

CAPÍTULO 2

ALGORITMOS GENÉTICOS

2.1 Introdução

Algoritmos Genéticos (*Genetic Algorithms* - GAs) são heurísticas que simulam o processo natural da evolução [5]. Pertencem à classe dos Algoritmos Evolutivos (*Evolutionary Algorithms* - EAs) [6] e são comumente utilizados para solucionar problemas de otimização, busca e aprendizagem de máquina [5, 7].

A heurística envolve, a partir de uma ou mais populações de indivíduos (representações de soluções) do problema, realizar a evolução, através de métodos de seleção, cruzamento e mutação de características dos indivíduos, até que uma função de aptidão ou função objetivo (*fitness function*) alcance um resultado suficientemente adequado ao domínio do problema [5]. O pseudocódigo é apresentado pelo Algoritmo 1.

Algoritmo 1 Um algoritmo genético padrão [5, 8, 9, 10].

1: $g \leftarrow 0$
2: **Gera** a população inicial $P(g)$
3: **Avalia** $P(g)$
4: **while** $g <$ número de gerações **do**
5: $\quad g \leftarrow g + 1$
6: $\quad$ **Seleciona** indivíduos de $P(g-1)$ e os copia para $P(g)$
7: $\quad$ Realiza a operação de **cruzamento** sobre pares de indivíduos em $P(g)$
8: $\quad$ Realiza a operação de **mutação** sobre indivíduos em $P(g)$
9: $\quad$ **Avalia** a nova população $P(g)$
10: **end while**
11: **Retorna** o melhor indivíduo

Há quatro diferenças primordiais entre GAs e outros procedimentos mais tradicionais de busca e otimização [8, 7]: o uso de uma codificação do conjunto de parâmetros ao invés dos parâmetros em si; busca através de vários pontos (indivíduos, soluções); uso de informações apenas da função objetivo e não de conhecimentos derivados ou auxiliares; e uso de regras de transição probabilísticas no lugar de determinísticas (uso de aleatoriedade).

2.2 Nomenclatura

Os algoritmos genéticos aplicam operações baseadas na natureza, portanto a nomenclatura utilizada na área é fortemente relacionada com os termos aplicados na biologia [11]. A

Tabela 2.1 resume o significado de cada um dos termos específicos.

Genótipo	Código adaptado para representar os parâmetros de um problema na forma de *string*.
Cromossomo	Uma *string* de parâmetros codificada (binário, ponto flutuante, etc.)
Indivíduo	Um ou mais cromossomos com um valor de aptidão associado.
Gene	A representação de um parâmetro do problema sendo resolvido.
Alelo	Valor no qual um gene pode assumir (binário, inteiro, real, etc.)
Locus	A posição que o gene ocupa no cromossomo.
Fenótipo	Versão do problema do genótipo (versão algorítmica), adequada para avaliação.
Aptidão	Número real indicando a qualidade de um indivíduo como uma solução do problema.
Ambiente	O problema. Ele é representado como uma função indicando a adequação dos fenótipos.
População	Um conjunto de indivíduos com suas estatísticas associadas (média de aptidão, distância Hamming, ...).
Seleção	Política para selecionar um indivíduo de uma população (seleção do mais apto, seleção por torneio, ...).
Cruzamento	Operação que mescla o genótipo de dois pais selecionados para fornecer dois novos filhos.
Mutação	Operação que espontaneamente muda um ou mais alelos de um genótipo.

Tabela 2.1: Nomenclatura utilizada pela área [11].

O algoritmo canônico opera em uma população de *strings* ou, em geral, estruturas complexas arbitrárias representando tentativas de solução. Na literatura da área, cada *string* é chamada de indivíduo e é composta por um ou mais cromossomos e um valor de aptidão. O cromossomo representa um conjunto de parâmetros chamados genes, com cada gene codificado, comumente, em binário [11].

Segundo Alba e Troya [11], a função de aptidão se comporta como o ambiente dentro do qual a versão decodificada do genótipo (o fenótipo) é avaliada.

2.3 Operadores e Parâmetros

A base de todos os algoritmos genéticos envolve pelo menos três operadores [12]: seleção, cruzamento e mutação.

A seleção, como o próprio nome indica, seleciona os cromossomos em uma população para reprodução. Quanto mais apto o cromossomo (isto é, segundo o resultado da função objetivo), mais chances ele tem de ser selecionado para reprodução [12], de acordo com o método usado. Essa reprodução pode ser tanto sexuada (cruzamento) quanto assexuada (mutação).

O cruzamento (*crossover*) é feito a partir de dois indivíduos da população. Em um dos métodos possíveis, o operador escolhe aleatoriamente uma posição de um cromossomo (*locus*) e troca as *substrings* antes e depois desse *locus*, criando dois descendentes, com uma parte do original e outra do segundo cromossomo e vice-versa. É semelhante ao processo biológico natural do cruzamento, onde o filho herda uma parte dos genes da mãe e outra do pai [12]. Além disso, o cruzamento possui uma taxa que indica as chances dele ocorrer entre dois indivíduos selecionados [9].

Na mutação ocorre uma troca aleatória de alguns bits em um cromossomo. A troca pode ocorrer em cada posição da *string* solução, de acordo com uma probabilidade normalmente muito baixa (taxa de mutação) [12]. Por mais que seja considerada uma operação secundária, a mutação é útil para escapar de mínimos (ou máximos) locais [9].

Como GAs trabalham com uma população de indivíduos, o tamanho da população acaba se tornando um parâmetro importante a ser cautelosamente estudado e escolhido. Um segundo parâmetro essencial, como indicado pelo Algoritmo 1, é a quantidade de gerações (iterações) que o algoritmo realizará antes de retornar a melhor solução encontrada. Em outras palavras, o segundo parâmetro é o critério de parada dos algoritmos genéticos.

Comumente realizada de forma aleatória, a geração da população inicial de cromossomos é uma outra operação responsável por influenciar a qualidade dos resultados encontrados [13]. Por exemplo, Benitez e Lopes [14] usaram uma estratégia retroativa (*backtracking strategy*) para gerar a população inicial, a fim de tanto corrigir certos cromossomos inválidos gerados aleatoriamente, quanto melhorar a qualidade dos indivíduos gerados.

Os algoritmos genéticos ainda podem operar utilizando uma estratégia elitista, onde os melhores indivíduos da geração anterior são mantidos vivos na nova população gerada, ao invés de apenas manter os filhos. Por mais que seja algo inexistente na natureza, isso garante que as soluções encontradas ao longo das gerações apenas tendam a melhorar. Todavia, isso pode acabar acelerando uma convergência para um mínimo local, o que, por sua vez, pode ser superada com uma alta taxa de mutação [5].

GAs funcionam tradicionalmente seguindo o modelo de evolução de Darwin [14]. Todavia, há dois outros modelos evolutivos utilizados para melhorar o desempenho dos algoritmos genéticos: a hipótese (descreditada, mas passível de aplicação aos GAs) de Lamarck, em que características adquiridas ao longa da vida de um indivíduo possam ser passadas aos filhos; e o efeito de Baldwin em que o aprendizado de um indivíduo ao longo de sua vida melhora sua aptidão (porém, ao contrário da hipótese de Lamarck, o genótipo não é modificado). Ambas as estratégias podem usar uma busca local para melhorar a aptidão dos cromossomos e a sobrevivência dos indivíduos [15].

2.4 Aplicações

Em [12] estão descritas algumas áreas e problemas em que GAs são empregados:

- Otimização - otimização numérica, disposição de circuito (*circuit layout*) e escalonamento por *job shop*.
- Programação automática - evoluir programas para tarefas específicas e projetar outras estruturas computacionais tais como autômatos celulares e ordenação de

redes.

- Aprendizagem de máquina - usados para diversas aplicações (como tarefas de classificação e previsão), além de aspectos particulares como pesos para redes neurais e sensores para robôs.
- Economia - modelar processos de inovação, o desenvolvimento de estratégias de licitação e o surgimento de marcados econômicos.
- Sistemas imunológicos - modelar aspectos da sistemas imunológicos naturais, incluindo mutação somática durante a vida de um indivíduo e a descoberta de famílias multigênicas ao longo da evolução.
- Ecologia - modelar fenômenos ecológicos como, por exemplo, corridas armamentistas biológicas, coevolução parasita-hospedeiro, simbioses e fluxo de recursos.
- Populações genéticas - estudar questões relacionadas, como "Sob quais circunstâncias um gene para cruzamento será evolutivamente viável?".
- Evolução e aprendizagem - usados para estudar como a aprendizagem de um indivíduo e evolução de espécies afetam um ao outro.
- Sistemas sociais - estudar a evolução da cooperação e comunicação entre sistemas multiagente.

CAPÍTULO 3

ALGORITMOS GENÉTICOS PARALELOS

3.1 Introdução

Algoritmos Genéticos Paralelos (*Parallel GAs* - PGAs) são variações dos GAs em alguns aspectos centrais aos quais a paralelização pode ser aplicada, possibilitando um rendimento maior, com resultados melhores e otimização do uso dos recursos computacionais, especialmente do tempo [7, 16].

Desde o início de sua utilização, diversos trabalhos em torno do tema foram desenvolvidos, trazendo uma ampla gama de soluções para o problema da implementação de PGAs (como programação baseada em GPUs) e sua otimização (como técnicas para aceleração superlinear - *superlinear speedup* [16]).

3.2 Taxonomia

Nowostawski e Poli propuseram em [17] uma taxonomia para algoritmos genéticos paralelos levando em consideração as seguintes características: como a aptidão é avaliada e como a mutação é aplicada; a quantidade de subpopulações (*demes*) utilizadas; se há várias subpopulações, como os indivíduos são permutados; como a seleção é aplicada (isto é, globalmente ou localmente).

Cada uma dessas características influencia em qual aspecto a paralelização é aplicada. De acordo com o foco dado, Nowostawski e Poli realizaram a seguinte classificação para PGAs em [17]:

- Paralelização através do modelo mestre-escravo (avaliação de *fitness* distribuída):
  - Síncrona;
  - Assíncrona;
- Subpopulações fixadas, com migração;
- *Demes* fixadas com sobreposição, porém sem migração;
- Algoritmos genéticos massivamente paralelos;
- *Demes* e sobreposição dinâmicas;
- PGAs estacionários (*steady-state*);

- PGAs bagunçados (*messy*);
- Métodos híbridos (por exemplo, subpopulações estáticas com migração e avaliação distribuída da aptidão dentro de cada subpopulação).

A ideia básica de qualquer programa paralelo é dividir o problema em tarefas menores e realizá-las usando diversos processadores [6, 18]. Essa abordagem (*divide-and-conquer*) pode ser aplicada de diferentes maneiras sobre os GAs. Uma classificação proposta por Cantú-Paz em [18] engloba três grandes características dos PGAs e uma combinação delas:

- Modelo mestre-escravo com uma população global;
- Uma população com paralelização *fine-grained*[1] (FGPGA, adequado para processamento massivamente paralelo, também é conhecido como GA celular, CGA);
- Várias populações com paralelização *coarse-grained*[2] (CGPGA, às vezes chamado de PGA modelo em ilha ou GA distribuído, DGA).
- Combinação de múltiplas *demes* (populações) com mestre-escravo ou GAs *fine-grained*.

Dentre esses quatro modelos, apenas o primeiro não altera o comportamento original do algoritmo.

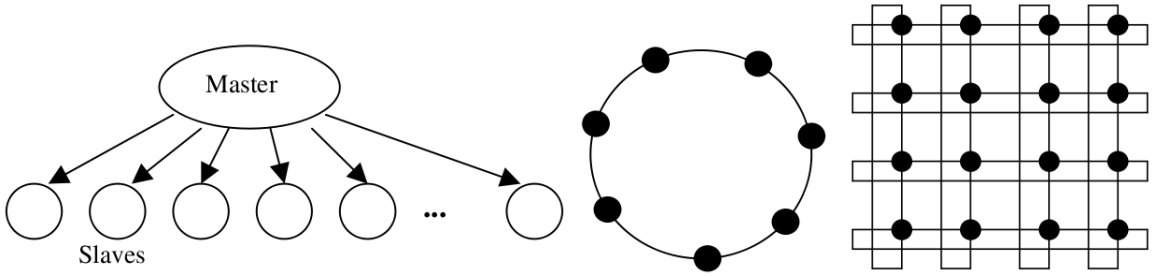
Em [11], além das três primeiras classes de [18], também há um quarto modelo de PGA retirado da taxonomia das técnicas de busca: o paralelismo automático (compilador). A Figura 3.1 fornece uma visualização de vários modelos.

3.2.1 Paralelização pelo Modelo Mestre-Escravo

O modelo de paralelização mestre-escravo, também chamado de paralelização global (*global* PGA, GPGA), utiliza uma única grande população enquanto avalia os indivíduos e aplica os operadores genéticos paralelamente [8]. Assim como em um GA sequencial (*Sequential or Standard Genetic Algorithm* - SGA), a seleção e a reprodução podem ocorrer entre quaisquer indivíduos sendo, portanto, globais. Em geral, esse método implementa programas no padrão mestre-escravo, em que o mestre armazena a população enquanto os escravos calculam o *fitness* (que é a operação mais comumente paralelizada) [19, 18, 20].

O algoritmo será síncrono se o mestre espera para receber todas as avaliações de *fitness* dos escravos. Desta forma, o PGA global carrega as mesmas características do SGA, tendo a velocidade como única diferença [19, 18, 11]. Segundo Cantú-Paz em [18], a maioria dos

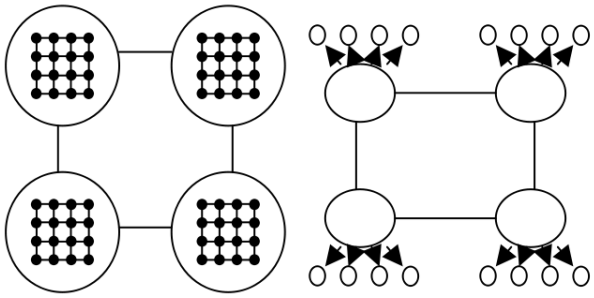
[1] Pequenos trechos de código para cada processador, com frequente comunicação entre as partes.
[2] Grande processamento local, com pouca comunicação entre as partes.



(a) Modelo mestre-escravo.

(b) Modelo CGPGA em anel.

(c) Modelo FGPGA em torus.



(d) Modelo combinado CGPGA com FGPGA.

(e) Modelo combinado CGPGA com mestre-escravo.

Figura 3.1: Exemplos de modelos de GAs paralelos [19].

GPGAs são síncronos e a maior parte da documentação científica assume que eles têm a mesma busca que os GAs simples (o que não ocorre com algoritmos assíncronos).

Este modelo não necessita de nenhuma arquitetura computacional específica. É viável implementá-lo em computadores tanto com memória compartilhada quanto distribuída [18].

A paralelização no modelo pode ocorrer também em operadores genéticos, como mutações e cruzamentos. Todavia, devido à simplicidade dessas operações, o *overhead*[3] da comunicação pode anular os ganhos de desempenho [18].

A grande vantagem dos PGAs mestre-escravo é a fácil implementação e o grande ganho no desempenho do algoritmo quando a avaliação precisa de considerável computação [19, 18]. Além disso, por manter as características dos SGAs (ao menos as versões síncronas), toda a teoria disponível sobre eles também é aplicável ao modelo global [6, 18].

3.2.2 Várias Populações com Paralelização *Coarse-Grained*

CGPGA é um método de paralelização frequentemente utilizado, existindo na literatura diversos artigos que descrevem inúmeros aspectos e detalhes de implementação. As características mais importantes desse modelo são o uso de algumas subpopulações relativamente grandes e migração [19, 18].

[3]Qualquer combinação de excesso no uso de recursos computacionais.

Outra característica, descrita por Alba e Troya [11], é a heterogeneidade de um CGPGA. Além da tradicional questão da homogeneidade do hardware, há a questão das buscas realizadas pelas ilhas. Isto é, as ilhas podem usar os mesmos operadores e taxas, ou cada uma pode carregar uma configuração distinta, portanto heterogênea. Essas distinções podem ser vantajosas quando comparadas com a um modelo homogêneo, pois elas podem ser ajustadas para focar na prospecção ou na exploração do espaço das soluções, dependendo do problema.

Segundo Cantú-Paz [18], alguns estudos confirmaram um princípio antigo em populações de seres vivos: traços genéticos favoráveis espalham-se mais rapidamente em populações menores. Todavia, também observou-se que a qualidade da solução (obtida nessas pequenas *demes*) foi menor, estabilizando em um *fitness* reduzido se comparado com as das grandes populações. Nessas análises também constatou-se a existência de uma taxa de migração crítica, abaixo da qual o desempenho do algoritmo é afetado pelo isolamento das *demes*, e acima da qual as subpopulações encontram uma solução de igual qualidade àquela encontrada pela população pan-mítica (uma única e grande população de indivíduos).

Esses estudos trouxeram questões importantes sobre PGAs que ainda não foram resolvidas [18]: Qual é o nível de comunicação necessário para fazer um PGA se comportar como um GA pan-mítico? Qual é o custo dessa comunicação? O custo dessa comunicação é pequeno o suficiente para tornar este modelo uma alternativa viável para o *design* de GAs paralelos?

Cantú-Paz [18] ainda formulou hipóteses sobre as razões para a popularidade dos algoritmos de múltiplas *demes*:

- GAs de várias populações são como uma "simples" extensão do SGA: utiliza-se alguns GAs convencionais, executando cada um em um nodo de um computador paralelo; em certos momentos pré-determinados, troca-se alguns indivíduos entre as subpopulações.

- O esforço para programar essas mudanças é relativamente pequeno, reaproveitando praticamente toda a versão sequencial, necessitando adicionar apenas poucas subrotinas para implementar a migração.

- Computadores paralelos *coarse-grained* estão bem disponíveis e, mesmo quando não estão, não é complicado simular um usando uma rede de nodos de processamento, ou ainda com apenas uma CPU e a ajuda de *software* livre (como MPI).

Existem várias aplicações para o modelo de PGAs de várias populações. Por exemplo, o modelo é amplamente utilizado no problema de particionamento de grafos, onde encontra-se soluções globais para problemas com mais de 40 mil variáveis de inteiros. Nos

testes, percebeu-se que o desempenho, medido como sendo a qualidade de uma solução e a iteração na qual foi encontrada, aumentou na mesma razão em que mais *demes* foram adicionadas [18].

Os algoritmos também são usados para encontrar soluções: ao problema de designação quadrática (*quadratic assignment problem*), que é outra aplicação de otimização combinatória; ao problema de distribuição de cargas computacionais (*distributing computing loads*) para processamento de nodos de computadores *Multiple Instruction, Multiple Data* (MIMD); e para a síntese de circuitos *Very-Large-Scale Integration* (VLSI); entre muitas outras [18].

3.2.2.1 Migração e Estudos Teóricos

No modelo CGPGA há uma operação adicional além da seleção, cruzamento e mutação do SGA: a chamada migração. Como o nome indica, essa operação é responsável por migrar indivíduos entre subpopulações diferentes, possibilitando a troca de informações (soluções) entre as diferentes *demes* [9]. Essa troca pode ser feita copiando-se um indivíduo mais apto de uma subpopulação para outra, ou realmente migrando-o [11].

Percebe-se rapidamente como PGAs podem ser mais complexos do que suas versões sequenciais quanto aos vários detalhes que afetam seus desempenhos [18]. Os CGPGAs possuem diversos novos parâmetros, como [19, 14, 8]: o número de subpopulações; o tamanho de cada uma delas; a topologia que define a comunicação entre as *demes* (como visto na Figura 3.1); a taxa de migração (fração de indivíduos de uma população que migram) normalmente variando entre 1% e 10% do tamanho da população [11]; a política para selecionar emigrantes; a política para substituir indivíduos antigos para receber imigrantes; e a frequência ou o intervalo (*gap*) de migração, que é o número de gerações entre migrações sucessivas [19, 14, 8].

Uma questão teórica importante relacionada esses novos parâmetros é determinar se, e sob quais circunstâncias, um CGPGA pode encontrar uma solução melhor que o GA tradicional. Duas importantes observações foram feitas nesse assunto. A primeira diz que subpopulações isoladas têm uma tendência a convergirem para diferentes soluções, enquanto migração e cruzamento podem combinar soluções parciais, o que indica que CGPGA com baixa taxa de migração pode encontrar soluções melhores para funções separadas. A segunda observação é que o cruzamento de soluções parciais resulta em indivíduos de pior *fitness*, o que indica que o GA sequencial possui uma vantagem [18].

Segundo consta em [18] e [11], a migração pode ocorrer de forma tradicionalmente síncrona, ou de forma assíncrona. No segundo caso, um dos algoritmos criados força a migração somente após as *demes* terem convergido completamente, com o propósito de restaurar a diversificação para prevenir a convergência global para uma solução de baixa qualidade. Essas avaliações logo trouxeram à tona outra importante questão [18]: quando

é o momento mais adequado para a migração ocorrer? Se a migração acontece cedo demais entende-se que a quantidade de *loci* adequados nos migrantes possa ser pequena demais para influenciar a busca a um caminho melhor, além de gastar caros recursos de comunicação

3.2.2.2 Topologias de Comunicação

A topologia utilizada em um PGA pode influenciar muito seu desempenho. Em uma densa conexão boas soluções são rapidamente espalhadas, enquanto em uma topologia mais esparsa, soluções demorarão a serem comunicadas, já que as subpopulações estarão mais isoladas umas das outras, permitindo a aparição de diferentes soluções, que podem ser mais tarde cruzadas formando indivíduos potencialmente mais adequados. Enquanto a primeira abordagem promove uma melhor mescla de indivíduos, ela implica em maiores custos de comunicação, dificultando a paralelização e escalabilidade dos DGAs [18, 11].

A topologia pode ser estática ou dinâmica. No segundo caso, uma *deme* não está restrita a se comunicar com um conjunto fixo de populações. Ao contrário, ela escolhe as populações mais adequadas segundo alguns critérios, como o impacto que o emigrante terá sobre elas medido através da avaliação da diversidade da subpopulação de destino ou da distância genotípica entre as duas populações (por exemplo, com a análise do melhor indivíduo para representante) [18].

A comunicação ainda pode ser classificada em modelo de ilhas isoladas e modelo de ilhas conectadas. Na primeira, a comunicação apenas ocorre no final da busca para selecionar a melhor solução No segundo, a comunicação entre as ilhas ocorre de acordo com o intervalo de migração [9].

Alguns exemplos de topologias de comunicação incluem malha (*mesh*) toroidal 4x4, conexão total, anel unidirecional, anel bidirecional, hipercubo 4-D [18], hipercubo 3-D [20] e topologias hierárquicas, como árvores [11].

3.2.3 PGAs *Fine-Grained*

CGAS possuem apenas uma população e uma estrutura que limita as interações entre indivíduos, os quais podem apenas competir e acasalar (seleção e cruzamento) com seus vizinhos. Como as vizinhanças se sobrepõem, boas soluções podem ser disseminadas por toda a população, provendo um mecanismo implícito de migração [19, 18, 8].

Ao lidar com modelos celulares, os principais fatores a serem levados em consideração são a estrutura da vizinhança, o esquema de seleção e o esquema de substituição [8]. Uma estrutura muito utilizada pelos pesquisadores é a grade 2-D, por sua semelhança com a topologia de muitos computadores paralelos [18].

Entre as diversas estruturas espaciais usadas para organizar a população, encontra-se

a adotada inicialmente por uma implementação de um PGA assíncrono chamada ASPARAGOS (*Asynchronous PGA*) [21]. Nela, a população está organizada em uma estrutura que se assemelha com uma escada, na qual os dois fins (primeiro e último degraus) estão grudados. Depois, uma estrutura linear foi escolhida e usada para fins de comparação [18].

Os FGPGAs também são aplicados para resolver alguns problemas difíceis. Entre eles, a aplicação sobre o problema de "embalar em uma caixa" (*bin packing problem*) bidimensional trouxe resultados satisfatórios. O modelo também foi aplicado ao problema de predição da estrutura secundária do RNA [18].

Segundo Pit [8], os resultados obtidos em diversas pesquisas indicam que os CGAs obtêm resultados melhores do que os SGAs, são menos suscetíveis à ficarem presos em mínimos locais, conseguem encontrar diversas soluções ótimas em uma mesma execução (ao longo das gerações), a diversidade dos genes é maior e são mais robustos quanto às configurações dos parâmetros.

3.2.4 Algoritmos Genéticos Paralelos Hierárquicos

Um PGA hierárquico (*Hierarchical* PGA - HPGA) é uma combinação de ao menos dois métodos para paralelizar GAs [6], buscando melhorar ainda mais o desempenho ao aproveitar as melhores características de cada um dos métodos. Grande parte delas aumenta a complexidade do cenário de PGAs, enquanto outras conseguem mantê-la [18].

Uma grande parcela dessas abordagens mescladas possui no nível superior algoritmos de múltiplas populações. Algumas têm no nível inferior um FGPGA. Uma hierarquia alternativa usa no nível inferior subpopulações espacialmente estruturadas. Isto é, as subpopulações estão conectadas por uma topologia em anel, enquanto cada uma delas está organizada como um toroide. Quando comparado com outros modelos, esse PGA hierárquico encontrou soluções melhores no geral [18].

Um exemplo de abordagens combinadas, desenvolvido para evitar a convergência prematura e manter a diversidade, é o modelo hierárquico de competição justa (*Hierarchical Fair Competition* - HFC) [10].

O *Hierarchical Fair Competition-based PGA* (HFC-PGA) é inspirado nas competições estratificadas que usualmente ocorrem na sociedade. Isto é, as camadas (neste caso, subpopulações), são divididas de acordo com a aptidão dos indivíduos, sendo que um indivíduo de uma *deme* de baixa *fitness* só poderá migrar para uma subpopulação de alta aptidão quando essa for superior ao limiar de aceitação da *deme* receptora [10].

Em seus testes, Lee *et al.* [10] mostraram que o HFC-PGA, quando comparado com o SGA, obteve um desvio médio menor (ou seja, maior precisão e proximidade em seus resultados), além de uma leve melhora no desempenho.

3.2.4.1 Exemplos e Aplicações

Lim *et al.* [6] apresentam um *Grid-Enabled HPGA Framework* (GE-HPGA), isto é, um conjunto de ferramentas com uma API[4] específica para auxiliar no desenvolvimento de HPGAs com suporte à computação em grade.

Com isso, Lim *et al.* [6] realizaram um estudo empírico usando um HPGA de dois níveis: no primeiro, tem-se o modelo de DGA, enquanto no segundo há um GPGA. Os resultados mostram que uma aceleração pode ser obtida desde que os limites no custa da função de aptidão (*fitness*), tamanho do *cluster* e *overheads* da comunicação sejam satisfeitos.

Wen-hua *et al.* [20] desenvolveram um HPGA, chamado HPGA heterogêneo assíncrono (*Asynchronous Heterogeneous HPGA* - AHHPGA), para resolver um *Redundancy Allocation Problem* (RAP), provadamente classificado como $\mathcal{NP}$-difícil. Esse modelo hierárquico utiliza um DGA no nível superior (em hipercubo tridimensional com dois grupos de nós com diferentes propriedades) e um CGA na camada inferior. Além disso, há três modelos de migração é assíncrona, garantindo o refinamento ou a expansão dos melhores resultados.

Wen-hua *et al.* [20] mostram que o AHHPGA resolve algumas dificuldades comuns em GAs, como a convergência prematura, enquanto aumenta a eficiência da heurística.

Benitez e Lopes em [14] descrevem um HPGA aplicado ao problema de predição de enovelamento de proteínas (*Protein Folding Prediction problem* - PFP), estruturado com um DGA e GPGA. Para resolver esse problema usa-se o modelo *3-Dimensional Hydrophobic-Polar Side-chain* (3DHP-Side-chain). Essa combinação (problema e modelo) é provada ser $\mathcal{NP}$-completa, sendo este o motivo para uso de heurísticas para resolvê-la, como GAs, que não apenas são adequadas, como têm-se mostrado eficientes para o PFP.

Benitez e Lopes [14] obtiveram melhores resultados em 7 de 10 casos com o seu HPGA, em comparação com os *benchmarks* de outro trabalho que usava um PGA não hierárquico para resolver o PFP.

[4]API (*Application Programming Interface*) é um conjunto de regras e especificações que *softwares* podem seguir para conseguir comunicar-se entre si. No caso do *framework*, permite que suas sub-rotinas sejam usadas por outros programas.

CAPÍTULO 4

JOB SHOP SCHEDULING

4.1 Introdução

Este capítulo trata sobre a definição do problema de *Job Shop Scheduling* (JSSP).

Escalonamento é a alocação de recursos compartilhados ao longo do tempo para atividades concorrentes. No JSSP, as máquinas são os recursos e os *jobs* representam as atividades. Cada máquina pode processar apenas um *job* por vez, sendo que cada etapa do processamento é chamada de operação [3].

JSS é um problema $\mathcal{NP}$-difícil a partir de $m \geq 3$ máquinas e tempo de processamento unitário das operações ($p_{ij} = 1$) [4].

4.2 Instância

Uma instância do problema de *Job Shop Scheduling* é dada pela tupla (J, O, M, P), onde (adaptado de Brucker [22] e Brucker e Knust [4]):

- J é o conjunto de trabalhos (*jobs*) $\{J_1, ..., J_n\}$, os quais cada trabalho J_i ($i \in [1..n]$) é composto por um conjunto finito de operações O_i. Isto é, $O_i = \{O_{i1}, ..., O_{in_i}\}$. Além disso, essas operações devem ser executadas nesta respectiva ordem $j \in [1..n_i]$. Em outras palavras, há uma restrição $O_{ij} \rightarrow O_{i,j+1}$ de precedência entre as operações de um mesmo trabalho, o que significa que a operação $O_{i,j+1}$ não pode ser iniciada antes da finalização de O_{ij}. Não há precedência entre operações de *jobs* diferentes;

- O é o conjunto dos conjuntos de operações de seus respectivos trabalhos J_i, $i \in [1..n]$. Ou seja, $O = \{O_i, ..., O_n\}$;

- M é o conjunto de máquinas $\{M_1, ..., M_m\}$ disponíveis para realizar os trabalhos. Há uma máquina $\mu_{ij} \in M$ e um tempo de processamento $p_{ij} \in \mathbb{N}^*$ associados a cada operação O_{ij}. Uma operação O_{ij} não pode ser interrompida ou dividida enquanto estiver sendo executada por uma máquina μ_{ij} (não há preempção).

- $P = (p_{ij})$ contém os tempos de execução de toda operação O_{ij} na respectiva máquina $\mu_{ij} \in M$, com $i \in [1..n]$ e $j \in [1..n_i]$.

Segundo Brucker e Knust em [4], pode-se assumir, sem perder a generalidade, que $\mu_{ij} \neq \mu_{i,j+1}$, caso contrário poder-se-ia substituir duas operações subsequentes por apenas uma a ser executada em uma máquina (esse problema também é conhecido como "*job shop*

sem repetição de máquinas"). Outro detalhe é que, "por padrão", haverá espaço suficiente no *buffer* entre as máquinas para armazenar um trabalho, caso ele tenha uma operação processada em uma máquina e a próxima ainda esteja ocupada com outro trabalho.

Dadas as diversas variações e restrições possíveis e estudadas tanto do JSSP quanto dos problemas de escalonamento em geral, utiliza-se uma classificação em três campos $\alpha|\beta|\gamma$ para facilitar a identificação do problema, descrevendo, respectivamente, o ambiente da máquina (e qual o tipo de escalonamento - no caso, *job shop*), as características do trabalho e o critério para minimização (ou função objetivo) [23]. Nesse sentido, um JSSP com 4 máquinas, tempo de execução igual a 2 unidades para todas as operações e buscando minimizar o *makespan* (descrito na seção seguinte) pode ser representado por: $J4|p_{ij} = 2|C_{max}$.

4.3 Resposta (Versão de Otimização)

A resposta é um escalonamento viável $S = (S_{ij})$, $i \in [1..n]$, $j \in [1..n_i]$, tal que:

- S_{ij} representa o tempo inicial da operação O_{ij};
- As restrições de precedência entre as operações $O_{ij} \rightarrow O_{i,j+1}$ são cumpridas;
- $S_{ij} + p_{ij} \leq S_{i,j+1}$ para todos os trabalhos J_i (a unidade de tempo em que a operação seguinte é iniciada é a soma do início da anterior com o tempo de processamento na respectiva máquina μ_{ij});
- O tempo total para realizar a atividade J_i mais demorada é chamado de *makespan*. Ele é representado por $C_{max} = max\{C_i | i \in [1..n]\}$, onde $C_i := S_{ij} + p_{ij}$ é o instante de tempo em que o *job* J_i é finalizado. O objetivo é minimizar o *makespan* que é, em outras palavras, o tempo para finalizar o último trabalho.

Para auxiliar a resolução do problema, define-se S_0 como sendo a atividade inicial ($S_0 = 0$) e S_{n+1} como sendo o trabalho final, ambos contendo (ou mesmo sendo) apenas uma operação sem custo de processamento (isto é, $p_0 = p_{n+1} = 0$ das respectivas operações $O_{0,1}$ e $O_{n+1,1}$), com $S_0 \rightarrow J_i \rightarrow S_{n+1}$ ($i \in [1..n]$). S_{n+1} representa, portanto, o *makespan* do escalonamento.

4.4 Pergunta (Versão de Decisão)

Dada uma instância (J, O, M, P) do JSSP mais um $K \in \mathbb{N}^*$, existe um escalonamento viável S cuja função objetivo é menor ou igual a K? (No caso do *makespan*, um escalonamento viável S tal que $C_{max}(S) \leq K$?)

4.5 Modelo de Grafo Disjuntivo

O modelo de grafo disjuntivo pode ser utilizado para representar escalonamentos para o JSSP [4].

Um grafo disjuntivo $G = (V, C, D)$ é um grafo com um conjunto de vértices V, um conjunto C de arestas direcionadas (conjunções) e um conjunto D de arestas não direcionadas (disjunções), tal que:

- V representa o conjunto de todas as operações (incluindo as operações auxiliares $O_{0,1}$ e $O_{n+1,1}$), que serão denotadas aqui por $v \in \mathbb{N}$ de tal forma que há uma enumeração iniciada em 0 para cada operação, na respectiva ordem $\{O_{0,1}, O_{1,1}, O_{1,2}, ..., O_{1,n_i}, O_{2,1}, ..., O_{n,n_i}\}$. Cada vértice v possui um peso correspondente ao tempo de processamento p_v (o qual seria o p_{ij} na máquina μ_{ij} da operação O_{ij});

- C representa as restrições de precedência entre as operações v de um mesmo trabalho J_i, com a adição das operações dos trabalhos auxiliares.

- D representa as diferentes ordens em que as operações em uma mesma máquina podem ser escalonadas. Consiste nas arestas não direcionadas entre pares de vértices cujas operações devem ser processadas pela mesma máquina. Isto é, para toda operação $a, b \in V$ com $\mu_a = \mu_b$ e $J_a \neq J_b$, há uma aresta não direcionada denotada por $a - b$ (significando que essa disjunção $a - b$ deve satisfazer $S_a + p_a \leq S_b$ ou $S_b + p_b \leq S_a$).

O problema para encontrar um escalonamento viável S para o JSSP é equivalente, destarte, a fixar uma direção para cada aresta em D de forma que o grafo resultante não contenha ciclos [4].

Um conjunto S de disjunções fixadas é chamado de seleção. Se para toda disjunção foi fixada uma direção, então a seleção é completa. Se o grafo $G(S) = (V, C \cup S)$ é acíclico, então S é consistente [4].

4.6 JSSP e Algoritmos Genéticos

Para problemas de otimização em geral, encontrar soluções ótimas não é "fácil". Isto ocorre justamente por boa parte deles pertencerem à classe $\mathcal{NP}$-difícil quando os parâmetros não são completamente controlados. No caso do JSSP, logo que aumenta-se a quantidade de máquinas ou trabalhos de 2 para 3, não há mais algoritmos (pseudo-)polinomiais capazes de resolver variações cada vez mais genéricas do problema. Para estes casos, o uso de heurísticas em aplicações reais é extremamente importante. Com o uso de estimativas pelas funções objetivos, é possível alcançar resultados satisfatórios

(não necessariamente ótimos, mas suficientemente bons quanto à relação tempo-custo). É nesse grupo que encontram-se os algoritmos genéticos [4].

4.6.1 Aplicação de GAs para o JSSP

Para aplicar GAs para resolver o JSSP, pode-se utilizar o modelo de grafo disjuntivo. Neste caso, as populações de indivíduos podem ser diversas seleções (completas e consistentes) representando escalonamentos para o JSSP em questão [4]. Cada um dos indivíduos deverá ser analisado por uma função objetivo (também relativa à função objetivo do JSSP) e passará pelos métodos de seleção, cruzamento e mutação, de tal forma que os piores resultados (pela função objetivo) serão "naturalmente" descartados e os melhores mantidos.

Ao gerar novas seleções S, um dos pontos mais importantes é verificar a viabilidade da seleção, ou seja, se o grafo $G(S)$ resultante é acíclico [4]. Ao saber que uma seleção é ilegal (não é viável), há duas maneiras de abordar esse problema, segundo Fleming *et al.* [3]: corrigir a seleção ou aplicar penalidades a ela.

No caso de corrigir uma seleção ilegal, há três operações básicas que podem ser feitas. A primeira é utilizar um algoritmo de harmonização (com a distância Hamming para averiguar a diferença entre dois indivíduos). A harmonização possui duas etapas centrais: harmonização local e global. A local é responsável por remover inconsistências de ordenação das operações dentro de uma máquina M_i, enquanto a global corrige inconsistências na ordenação das máquinas [3].

Essas reparações buscam apenas analisar a aptidão do indivíduo, sendo a etapa de verdadeira substituição de partes do indivíduo apenas opcional, de forma a garantir a diversidade da população [3].

Em testes de GAs e variações apresentadas por Fleming *et al.* [3] (mais especificamente, o *Multi-Step Crossover Fusion GA* - MSXF-GA), para problemas como $J10|n = 10|$, $J20|n = 10|$ e $J15|n = 15|$ (um total de 10 versões $\mathcal{NP}$-difíceis do JSSP utilizadas para *benchmarking*), com 30 execuções, o MSXF-GA encontrou ótimos resultados, com pouca diferença entre os melhores e os piores casos. A heurística inclusive encontrou, nos melhores casos, a solução ótima para metade dos 10 problemas. Além disso, quatro desses escalonamentos ótimos foram encontrados rapidamente, em termos comparativos com outros algoritmos.

Não apenas usados separadamente, mas quando combinados com outras heurísticas ou algoritmos determinísticos, os GAs são capazes de otimizar (e serem otimizados por) diversos métodos, trazendo benefícios e possibilitando encontrar até mesmo soluções ótimas com um menor custo computacional [3].

CAPÍTULO 5

EXPERIMENTOS

5.1 Introdução

Neste capítulo são detalhados todos os experimentos realizados, desde etapas do desenvolvimento dos algoritmos até os parâmetros nas diversas implementações. Os algoritmos foram escritos na linguagem de programação C e estão disponíveis para acesso em `https://github.com/Erik-Pucci/GAs-applied-to-JSSPs`.

Para desenvolver esses algoritmos foi-se utilizada a biblioteca livre GAUL [24] versão 0.1850-0, de licença GNU GPL versão 2. Para os gráficos, foi utilizado o *software* Gnuplot versão 4.4. Ademais, todas as execuções foram feitas em um computador com um processador modelo "Intel(R) Core(TM) 2 Duo CPU P8600", de 2,4 GHz, e memória RAM de 4 GiB.

A Seção 5.2 descreve a representação utilizada para instanciar o JSSP. Em seguida, a Seção 5.3 apresenta em detalhes quais operadores foram experimentados, incluindo desde a geração da população inicial até as mutações. A Seção 5.4 trata dos parâmetros e operadores usados em cada caso de teste. A Seção 5.5 informa os testes usando o modelo em ilha (versão sequencial, não paralela). Finalmente, a Seção 5.6 detalha os experimentos entre versões do modelo GA tradicional com o modelo em ilha (também sequencial).

5.2 Representação

Antes de mais nada, é necessário determinar como será feita a representação cromossômica das instâncias do JSSP [13]. Em [25], são descritas duas abordagens para solucionar esse problema. A primeira, conhecida por representação indireta, é quando codificam-se as instruções necessárias para um construtor de escalonamentos (*schedule builder*) [25]. Entre as possibilidades, há a *operation-based representation*, *job-based representation*, *preference-list-based representation*, *job-pair-relation-based representation*, *disjunctive-graph-based representation*, *completion-time-based representation*, *machine-based representation*, *random key representation* e a *priority-rule-based representation* [26].

A segunda, representação direta, é uma abordagem que procura codificar o próprio escalonamento. Alguns codificam os tempos de finalização das operações, outros os tempos iniciais das operações. Neste tipo de representação, nem todos os cromossomos correspondem à escalonamentos válidos, o que torna necessário o uso de métodos de reparamento (*repair methods*) ou funções de penalidade (*penalty functions*). No entanto, o uso dessas funções é ineficiente para JSSPs, pois a quantidade de escalonamentos possíveis é muito

maior do que a de escalonamentos válidos.

A representação escolhida foi a baseada em operações (*operation-based*). Nela, o cromossomo é uma sequência de inteiros representando a enumeração dos *jobs*, de 0 a $n-1$, onde n é a quantidade de trabalhos. Por exemplo, o cromossomo [1 3 2 1 2 3 2 1 3] para o problema exemplo descrito pela Tabela 5.1, constitui uma possível representação para o escalonamento da Figura 5.1. Cada trabalho aparece exatamente 3 vezes (pois possui 3 operações). Pela restrição da ordem das operações, a primeira vez que o *job* 1 aparece representa a sua respectiva primeira operação. A segunda vez, a segunda, enquanto a terceira representa a terceira operação. Assim acontece para todos os *jobs*. Se o um trabalho é repetido uma quantidade de vezes igual ao número de operações, então o cromossomo sempre representa uma solução viável para o problema [13].

Trabalho	Máquina (Tempo de Processamento)		
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

Tabela 5.1: Um problema exemplo $J3|n = 3|$ (3 trabalhos e 3 máquinas) [27].

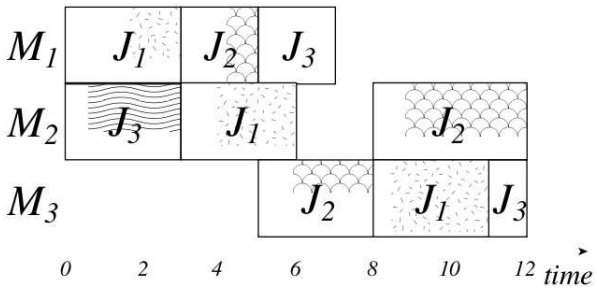


Figura 5.1: Gráfico de Gantt representando uma solução para o problema $J3|n = 3|$ [27].

5.3 Operadores

5.3.1 Função Objetivo

A função objetivo ou função de aptidão (*fitness function*) usada calcula o *makespan* de cada cromossomo. Para isso, ela percorre os genes da esquerda para a direita no cromossomo, calculando o tempo de finalização de cada operação. A partir desses tempos, obtém-se o maior entre os das últimas operações, que representa o *makespan*, já que as operações devem ser executadas na respectiva ordem no problema de JSS.

5.3.2 População Inicial

A função de geração da população inicial de indivíduos trabalha de maneira (pseudo)aleatória: os cromossomos são construídos (pseudo)aleatoriamente, mas de forma

que representem um escalonamento viável para o problema em questão, através do uso de um vetor auxiliar contendo a quantidade de operações de cada trabalho já inseridas no cromossomo. À medida que um trabalho já está completo (número de operações é igual ao número de máquinas), são gerados genes aleatórios até que a última operação de todos os *jobs* tenha sido inserida no cromossomo.

Outra possibilidade que mostrou-se promissora é o uso de uma adaptação do algoritmo de Giffler & Thompson (G&T *algorithm*) apresentada por Park *et al.* em [13]. Isso possibilita a geração de uma população inicial diversificada e de qualidade (isto é, com um bom *fitness*). O Algoritmo 2 mostra as etapas desse processo, apenas com o Passo 3 modificado (modificado $r_{im*} < t(C)$ por $r_{im*} \leq t(C)$).

Algoritmo 2 Descrição da geração da população inicial baseado no algoritmo de G&T [13].

1: **Passo 1:** Seja C o conjunto com as primeiras operações de cada trabalho. Atribua $r_{ij} = 0$ para todas as operações O_{ij} em C, onde r_{ij} é o menor tempo (*earliest time*) em que que a operação O_{ij} pode ser executada.
2: **Passo 2:** Calcula $t(C) = min_{O_{ij} \in C}\{r_{ij}\}$. Seja $m*$ a máquina onde o mínimo ocorre.
3: **Passo 3:** Seja G o conjunto de conflitos de todas as operações O_{im*} da máquina $m*$ tais que $r_{im*} \leq t(C)$.
4: **Passo 4:** Seleciona aleatoriamente uma operação de G e a escalona.
5: **Passo 5:** Remove de C a operação escalonada e inclui sua sucessora imediata (do mesmo *job*) em C, caso haja.
6: **Passo 6:** Atualiza os valores de r_{ij} em C e retorna para o Passo 2 até que todas as operações tenham sido escalonadas (e C torne-se vazio).

5.3.3 Seleção

Para a operação de seleção, foram aproveitados os procedimentos já implementados pela biblioteca GAUL [24], com exceção do "Torneio de 2, 0.75", retirado de Park *et al.* [13]. São eles:

- Amostragem Universal Estocástica (*Stochastic Universal Sampling* - SUS) - seleciona os indivíduos através da SUS.
- SUS quadrada - seleciona os indivíduos usando SUS com o quadrado do *fitness*.
- Mais apto - seleciona o indivíduo mais apto e um segundo aleatório.
- Torneio de 2 - seleciona o melhor através de um torneio entre dois indivíduos selecionados aleatoriamente.

- Torneio de 3 - idem ao torneio de 2, porém com 3 indivíduos selecionados aleatoriamente.
- Torneio de 2, 0.75 - seleciona o melhor com probabilidade de 0.75 através de um torneio entre dois indivíduos selecionados aleatoriamente.
- Agressivo - seleciona o primeiro indivíduo aleatoriamente, onde a probabilidade é uma função exponencial de classificação, e o segundo aleatoriamente. Este é o único método em que os indivíduos podem ser os mesmos.
- Classificação linear - seleciona de tal forma que a probabilidade é baseada em uma função linear de classificação.
- Aleatório - seleciona aleatoriamente.
- Aleatório por classificação - seleciona o primeiro indivíduo aleatoriamente onde a probabilidade é uma função linear de classificação, e o segundo sequencialmente.
- Round-Robin (RR) - seleciona um indivíduo de acordo baseado nesse algoritmo para a operação de mutação e dois indivíduos através da seleção por SUS para o cruzamento.

Para o cruzamento, dois indivíduos diferentes são selecionados. Para a mutação, apenas um é selecionado, de acordo com a primeira descrição de cada seleção.

5.3.4 Cruzamento

Quatro cruzamentos foram implementados. O primeiro deles é uma modificação do *Precedence Preservative Crossover* (PPX) [28], chamado aqui de PPX modificado (PPXm). Nele, um vetor auxiliar, gerado aleatoriamente, contendo elementos do conjunto $\{1, 2\}$ define a origem dos genes para cada *loci* dos pais $P1$ e $P2$, respectivamente. Todos os genes de $P1$ são removidos de $P2$, respeitando a ordem de ocorrência (pois ela define qual operação de um certo *job* deve ser removida). O cromossomo filho é iniciado vazio. Então, os genes são copiados para o filho segundo a origem determinada pelo vetor auxiliar. Se for de $P1$, apenas copia-se o gene do mesmo locus de origem. Se for de $P2$, copia-se seguindo a ordem em que os genes não removidos aparecem. Isso é feito até o cromossomo filho estar completo (e os pais "vazios") e o processo é repetido trocando-se a ordem dos pais, a fim de sempre gerar dois filhos. Um exemplo deste cruzamento pode ser visualizado na Tabela 5.2.

O segundo é o *Generalized Partially Mapped Crossover* (GPMX), também de Bierwirth *et al.* [28]. Neste cruzamento, uma *substring* de $P1$, também chamado de doador, é aleatoriamente escolhida (isto é, um *locus* representando o início da *substring* e um tamanho

são escolhidos aleatoriamente, com o último variando entre um terço à metade do comprimento do cromossomo). Os respectivos genes da cadeia de operações escolhida são removidos de $P2$ (receptor). Copia-se os genes do doador para o filho na mesma ordem e *loci* em que ocorrem em $P1$. O restante do cromossomo é então preenchido pelos genes que sobraram do receptor, mantendo a mesma ordem em que eles ocorrem em $P2$. O processo é repetido trocando-se $P1$ com $P2$ e usando o mesmo *loci* para a nova *substring*. Um exemplo do método pode ser visualizado na Tabela 5.3.

Indivíduo	Cromossomo
Pai (P1)	3 2 2 2 3 1 1 1 3
Mãe	1 ~~1~~ ~~3~~ ~~2~~ 2 ~~1~~ 2 3 ~~3~~
Origem dos genes	1 1 2 2 2 2 1 1 1
Filho PPXm	3 2 1 2 2 3 1 1 3

Tabela 5.2: Exemplo de uma operação do método PPXm para gerar o primeiro filho.

Indivíduo	Cromossomo
Pai (P1)	3 2 2 2 3 1 1 1 3
Mãe	~~1~~ 1 3 2 ~~2~~ 1 ~~2 3~~ 3
Filho GPMX	1 3 2 2 3 1 2 1 3

Tabela 5.3: Exemplo de uma operação do método GPXM para gerar o primeiro filho [28].

O terceiro é o chamado "*crossover* 4", apresentado por Park *et al.* [13] e baseado no GPMX. A diferença deste para o GPMX é que os genes de $P2$ (dos que sobraram após a remoção) cujo *locus* é anterior ao do primeiro *locus* da *substring* escolhida são adicionados ao filho. Em seguida, a *substring* de $P1$ é copiada para o filho a partir da posição livre seguinte. O resto dos genes de $P2$ são então copiados completando o filho. Como nos outros, o mesmo processo é feito para gerar um segundo filho, apenas trocando-se a ordem dos pais. Esse processo pode ser melhor compreendido com o exemplo apresentado pela Tabela 5.4.

Indivíduo	Cromossomo
Pai (P1)	3 2 2 2 3 1 1 1 3
Mãe	~~1~~ 1 3 2 ~~2~~ 1 ~~2 3~~ 3
Filho	1 2 2 3 1 3 2 1 3

Tabela 5.4: Exemplo de uma operação do método "*crossover* 4" para gerar o primeiro filho.

O último é o *Giffler-Thompson Focused* (GTF) *crossover* [26]. A operação é descrita pelo Algoritmo 3. A única diferença do original está no Passo 2, em que o melhor locus é escolhido aleatoriamente entre os elementos que possuem o maior conjunto de conflitos (*Conflict Set* - CS), que são todos os genes até o primeiro *locus* que possui a última operação de algum trabalho. O tamanho desse conjunto é, portanto, sempre igual ao

número de *jobs*. Por fim, o processo é repetido trocando-se $P1$ com $P2$ para gerar um segundo filho.

Algoritmo 3 Descrição do cruzamento GTF [26].

1: **Passo 1:** Escolha o primeiro dos pais $P1$.
2: **Passo 2:** Escolha o gene com o maior CS em $P1$. Seja $l*$ o *locus* desse gene, $CS*$ o CS e $|CS*|$ a quantidade de operações em $CS*$.
3: **Passo 3:** Gere $|CS*|$ filhos intermediários. A primeira parte desses filhos são os genes anteriores a $l*$ em $P1$.
4: **Passo 4:** Para os $|CS*|$ filhos, a operação (gene) na posição $l*$ são as operações em $CS*$ (uma operação para cada filho).
5: **Passo 5:** O restante dos genes são retirados de $P2$ na mesma ordem em que aparecem em $P2$, após as operações usadas em $P1$ mais a operação de cada filho intermediário tenham sido removidas. Logo, o restante de $P2$ acaba potencialmente diferente para cada filho gerado. Além disso, $P2 \neq P1$.
6: **Passo 6:** Retorna o melhor cromossomo dos $|CS*|$ filhos intermediários.

O funcionamento do cruzamento GTF pode ser acompanhado abaixo (para o primeiro filho):

Passo 1:

$P1 = 3\ 2\ 2\ 2\ 3\ 1\ 1\ 1\ 3$

$P2 = 1\ 1\ 3\ 2\ 2\ 1\ 2\ 3\ 3$

Passo 2:

O *locus* máximo é a posição 4. Aleatoriamente, $l* = 3$ (segundo gene 2), com $|CS*| = 3$ e $CS* = \{1, 2, 3\}$. **Passo 3:**

$F1$, $F2$ e $F3 = 3\ 2$

Passo 4:

$F1 = 3\ 2\ 1$

$F2 = 3\ 2\ 2$

$F3 = 3\ 2\ 3$

Passo 5:

$P2 = \cancel{1}\ 1\ \cancel{3\ 2}\ 2\ 1\ 2\ 3\ 3$

$F1 = 3\ 2\ 1\ 1\ 2\ 1\ 2\ 3\ 3$

$P2 = 1\ 1\ \cancel{3\ 2\ 2}\ 1\ 2\ 3\ 3$

$F2 = 3\ 2\ 2\ 1\ 1\ 1\ 2\ 3\ 3$

$P2 = 1\ 1\ \cancel{3\ 2}\ 2\ 1\ 2\ \cancel{3}\ 3$

$F3 = 3\ 2\ 3\ 1\ 1\ 2\ 1\ 2\ 3$

Passo 6:

Retorna o melhor entre os filhos $F1$, $F2$ e $F3$.

5.3.5 Mutação

A operação de mutação é importante para manter a diversidade populacional e evitar mínimos locais, por mais que muitas vezes seja considerada um mecanismo secundário de GAs, dada sua baixa taxa de ocorrência em populações naturais [8].

Entre as diferentes operações de mutação para o JSSP, Abdelmaguid cita em [29] as seguintes mutações:

- Por troca (*swap mutation* ou *reciprocal exchange mutation*) - troca o valor atribuído a dois genes diferentes selecionados aleatoriamente;
- Por inversão (*inversion mutation*) - inverte a ordem dos valores atribuídos para o conjunto de genes dispostos entre dois *loci* escolhidos aleatoriamente;
- Por inserção (*insertion* ou *shift mutation*) - seleciona aleatoriamente dois loci. O segundo recebe o gene do primeiro, enquanto o valor de todos os genes entre esses dois loci (incluindo o primeiro gene que estava no primeiro locus) são deslocados;
- Por deslocamento (*displacement mutation*) - é uma outra versão da *shift mutation* em que uma substring de genes, ao invés de apenas um gene, é movida para uma nova localização aleatória.
- Por vizinhança (*neighborhood search-based mutation*) - três genes diferentes são selecionados e são permutados de todas as maneiras possíveis. Dos cromossomos resultantes, diferentes do original (5 casos), o melhor é escolhido.

A diferença dessas mutações para a tradicional em uma cadeia de bits é a capacidade de criar novos indivíduos enquanto mantém um escalonamento viável para o problema de JSS.

Todas essas cinco abordagens foram implementadas e comparadas neste trabalho, com a adição de uma mutação por troca modificada, em que uma sequência de genes a partir de um certo locus é respectivamente trocada com uma sequência de mesmo tamanho a partir de um segundo locus, diferente do primeiro (chamada aqui de mutação por troca em multiponto ou *multi-point swap mutation*).

5.3.6 Elitismo

A biblioteca GAUL fornece cinco abordagens para trabalhar com elitismo. A primeira é simplesmente a ausência de qualquer elitismo, o que significa que os pais não passam para a geração seguinte (denominado aqui por "nenhum"). No segundo modelo, apenas o mais apto indivíduo sobrevive para a próxima geração, substituindo o menos apto dos filhos da geração anterior (chamado de "mais apto"). No terceiro, todos os pais aptos passam

para a geração seguinte ("total"). A quarta abordagem mantém os melhores indivíduos quanto à todas as características ("melhores"). Por último, há o conjunto de Pareto que mantém todos os indivíduos não dominantes, ou seja, que não são melhores em todas as características ("Pareto").

5.4 Configurações e Experimentos

5.4.1 Parâmetros e Operadores Base

Para os testes, a não ser que explicitado, os seguintes parâmetros e operadores foram utilizadas:

- População inicial - gerada aleatoriamente. O tamanho utilizado na literatura varia entre 10 [27], 50 [30, 25, 31], 100 [25, 26, 32, 33], 200 [13], 300 [32], 500 [25, 34, 35] e 600 [36], ou de acordo com o problema (por exemplo, o dobro do número de operações) [5]. A quantidade foi fixada em 100.
- Quantidade de gerações - entre 50 [31], 100 [13, 25], 150 [26], 300 [35], 400 [5] e 500 [34], ou dependente do tamanho do problema, como em [30] (50 vezes o número de *jobs*). A quantidade foi fixada em 100 gerações.
- Função objetivo - cálculo do *makespan*.
- Seleção - usada a seleção por SUS.
- Cruzamento - usado o PPXm. A taxa usada em diversos artigos da área varia entre 0.1 [31], 0.5 [27], 0.6 [25, 30, 35], 0.7 [13, 32, 33, 5], 0.8 [26] e 1.0 [34]. A taxa foi fixada em 0.7.
- Mutação - utilizada a mutação por troca (*swap*). A taxa usada para essa operação é normalmente baixa devido à raridade desse acontecimento em meio natural [8]. Ela varia de 0.01 [32, 33, 37, 38], 0.02 [34], 0.1 [13, 27, 25, 30, 38] à 0.2 [26, 5]. A taxa foi fixada em 0.1.
- Estratégia elitista: "nenhum".
- Usado o modelo de evolução de Darwin.
- Número de execuções independentes - o programa foi executado 100 vezes para uma análise mais confiável.
- A semente usada para gerar a sequência de números pseudoaleatórios utilizados pelas funções implementadas e de GAUL foi obtida a partir do tempo do sistema, através da função *time* da *GNU C Library*.

5.4.2 Problemas Usados

Os seguintes *benchmarks* foram usados, retirados da *OR-Library* [39], por acesso via *Internet*, e dispostos em ordem crescente do total de operações:

- 3 problemas, com parâmetros $J6|n = 6|$, $J10|n = 10|$ e $J5|n = 20|$ (5 máquinas e 20 *jobs*), de Fisher e Thompson (ft06, ft10 e ft20), também conhecidos por Muth e Thompson (mt06, mt10 e mt20) [40]. Os valores ótimos para esses problemas são, respectivamente, 55, 930 e 1165.
- $J15|n = 20|$ de Adams *et al.* (abz7) [41], de ótimo desconhecido. Park *et al.* [13] chegou em 685.
- $J10|n = 30|$ de Lawrence (la31) [42], de ótimo 1784.
- $J20|n = 20|$ de Yamada e Nakano (yn1) [38]. Park *et al.* [13] chegou em 925.

5.4.3 Tamanho da População

Para melhor verificar a relação do tamanho da população com o tamanho do problema e a qualidade da solução, foram testados os tamanhos 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 450 e 500. Outrossim, os mesmos testes foram feitos utilizando a abordagem elitista "total".

5.4.4 Quantidade de Gerações

Os mesmos parâmetros usados nos testes do tamanho da população foram usados para verificar a qualidade da solução de acordo com a variação da quantidade de gerações. As únicas diferenças foram o tamanho da população, que foi fixado em 100, e o foco dos testes, que é a variação do número de gerações entre 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 450 e 500. A fim de avaliar melhor essa variação, os mesmos testes foram feitos adotando a abordagem elitista "total".

5.4.5 Taxa de Cruzamento

Com os mesmos parâmetros dos testes anteriores (tamanho da população em e número de gerações em 100), foram testadas as seguintes taxas de cruzamento: 0.0 (nenhum cruzamento), 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 e 1.0 (o cruzamento sempre ocorre). A mutação continua em 0.1 e o número de execuções independentes em 100.

5.4.6 Taxa de Mutação

Todos os parâmetros estarão fixos, com exceção da taxa de mutação, que foi variado entre: 0.0 (nenhuma mutação), 0.001, 0.005, 0.01, 0.02, 0.04, 0.05, 0.07, 0.09, 0.1, 0.15 e 0.2.

5.4.7 Alternativa para a Geração da População Inicial

Duas abordagens para a geração da população inicial foram comparadas: a aleatória e a baseada no algoritmo de G&T [13]. Os parâmetros do tamanho da população, número de gerações, taxa de cruzamento, taxa de mutação e número de execuções independentes estarão fixas em: 100, 100, 0.7, 0.1 e 100. A função de aptidão continua sendo o cálculo do *makespan*, enquanto a mutação é por troca (*swap mutation*) e o cruzamento é o "*crossover* 4".

5.4.8 Alternativas para a Seleção

Todas as seleções descritas na Seção 5.3.3 foram testadas e comparadas. Os parâmetros continuam fixos, como descritos nas seções anteriores.

5.4.9 Alternativa para o Cruzamento

No mesmo modelo dos testes das abordagens para a geração da população inicial, foram comparados os quatro algoritmos previamente citados para a operação de cruzamento: PPXm, GPMX, "*crossover* 4" e GTF.

5.4.10 Alternativas para a Mutação

Como os testes das alternativas para a operação de cruzamento, foram comparadas as mutações por: troca, troca em multiponto, inversão, inserção, deslocamento e vizinhança.

5.4.11 Estratégias Elitistas

Três dos cinco modelos já disponibilizados pela biblioteca GAUL foram testados e seus resultados comparados: nenhum, mais apto e total. Os modelos "melhores" e "Pareto" funcionam como o "total" na ausência de múltiplos cromossomos ou características. Portanto, esses dois últimos modelos não foram averiguados.

5.5 Experimentos com o Modelo em Ilha

O modelo em ilha, como explicado na Seção 3.2.2, é composto por mais de uma população, organizadas em um anel unidirecional, e possui como parâmetros adicionais a taxa de

migração e o intervalo de migração. As duas únicas diferenças encontram-se no fato da taxa de migração funcionar como a probabilidade de um indivíduo migrar e no fato do algoritmo ter sido implementado sequencialmente, focando a avaliação nas soluções e não no tempo de execução (mais próximo do modelo síncrono da versão paralela). O modelo foi então executado da mesma forma que os experimentos anteriores, em seis casos de testes diferentes.

O primeiro varia a quantidade de populações em 2, 3, 4, 5, 6, 7, 8, 9, 10, 15 e 20, com a taxa de migração fixada em 0.001 e o tamanho da população fixo em 20 indivíduos. A biblioteca GAUL não implementa por padrão o parâmetro *gap* de migração, todavia funciona como se esse possuísse o valor 1, o que significa que a migração ocorre em toda geração.

O segundo caso de teste fixa a quantidade de subpopulações em 5, todas de tamanho 20, e varia a taxa de migração em 0.0 (nenhuma migração), 0.001, 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075 e 0.1.

O terceiro é uma modificação do segundo, executando os mesmos testes com uma população de tamanho 100.

Para analisar como o intervalo de migração pode influenciar os resultados, esse mecanismo foi implementado e testado variando a ocorrência da migração a cada 0 (ilhas isoladas), 1 (toda geração), 2, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90 e 100 gerações. A taxa de migração foi deixada em 0.001, enquanto foram mantidas 5 *demes* de 20 elementos e realizadas 500 gerações.

Para o quinto caso de teste, foram realizados praticamente os mesmos experimentos que o terceiro caso, com a taxa de migração modificada para 0.1 e variando o *gap* em 0, 1, 5, 10, 25, 50, 75 e 100.

Por fim, o sexto caso avalia o impacto da variação do intervalo de migração em 0, 1, 5, 10, 25, 50, 75 e 100. A taxa de migração ficou em 0.05, o número de gerações em 500 e o tamanho de cada uma das 5 subpopulações fixado em 100. Todavia, foi-se testado apenas para os problemas ft06, ft10 e ft20.

5.6 Comparação entre SGA e DGA

Para finalizar os experimentos a agregar os resultados finais, foram realizados testes comparativos entre as seguintes configurações de SGAs e DGAs (modelo em ilha, porém implementado sequencialmente):

- SGA básico (SGA1) - GA executado com as configurações padrões descritas na Seção 5.4;
- SGA com os melhores parâmetros (SGA2) - Configurações padrões, exceto o tamanho da população (diminuído para 20), a taxa de cruzamento (1.0), a taxa de

mutação (0.2) e a quantidade de gerações para cada execução (500);

- SGA com os melhores operadores (SGA3) - Configurações padrões, exceto a geração da população inicial (G&T), seleção (Torneio de 3), mutação (por inserção), cruzamento (GPMX) e elitismo "total";
- SGA com os melhores operadores e parâmetros (SGA4) - junta as modificações dos parâmetros e operadores respectivamente descritas para o segundo e o terceiro casos de teste com o SGA.
- Ilha básico (DGA1) - 20 indivíduos em cada ilha, 5 subpopulações, taxa de migração de 0.001 e intervalo igual a 1 (a migração ocorre em toda geração);
- Ilha com os melhores parâmetros (DGA2) - 20 indivíduos em cada ilha, taxa de cruzamento de 1.0, taxa de mutação de 0.2, 500 gerações, 20 subpopulações, taxa de migração de 0.05 e intervalo de 1 geração;
- Ilha com os melhores operadores (DGA3) - mesmos parâmetros do caso "ilha básico", com a geração da população inicial através do algoritmo adaptado de G&T, seleção por Torneio de 3, mutação por inserção, cruzamento com GPMX e elitismo "total";
- Ilha com os melhores operadores e parâmetros (DGA4) - Agregação dos parâmetros e operadores dos dois últimos experimentos com o modelo em ilha;
- Ilha com heterogeneidade (DGA5) - cada ilha possui uma configuração própria. São 7 ilhas em anel unidirecional, 300 gerações, elitismo "total" e intervalo de migração igual a 1 (toda geração) com:
  - Ilha 1: População inicial aleatória, seleção por Torneio de 3, mutação por inserção, cruzamento PPXm, 40 indivíduos, taxa de cruzamento de 0.7, taxa de mutação de 0.1 e taxa de migração de 0.05;
  - Ilha 2: População inicial aleatória, seleção por Torneio de 3, mutação por troca, cruzamento GPMX, 20 indivíduos, taxa de cruzamento de 0.8, taxa de mutação de 0.1 e taxa de migração de 0.075;
  - Ilha 3: População inicial G&T, seleção por Torneio de 3, mutação por inserção, cruzamento GPMX, 50 indivíduos, taxa de cruzamento de 0.7, taxa de mutação de 0.1 e taxa de migração de 0.1;
  - Ilha 4: População inicial G&T, seleção por Torneio de 3, mutação por vizinhança, cruzamento "*crossover* 4", 20 indivíduos, taxa de cruzamento de 0.9, taxa de mutação de 0.2 e taxa de migração de 0.05;

- Ilha 5: População inicial aleatória, seleção aleatória por classificação, mutação por deslocamento, cruzamento GPMX, 30 indivíduos, taxa de cruzamento de 0.7, taxa de mutação de 0.15 e taxa de migração de 0.075;
- Ilha 6: População inicial G&T, seleção por classificação linear, mutação por troca, cruzamento GTF, 20 indivíduos, taxa de cruzamento de 1.0, taxa de mutação de 0.1 e taxa de migração de 0.075;
- Ilha 7: População inicial aleatória, seleção por Torneio de 2, mutação por vizinhança, cruzamento GTF, 40 indivíduos, taxa de cruzamento de 0.6, taxa de mutação de 0.2 e taxa de migração de 0.075;

Todas essas versões seguiram os valores padrões que estejam ausentes nas descrições destes experimentos, conforme descrito pela Seção 5.4.

CAPÍTULO 6

RESULTADOS

6.1 Introdução

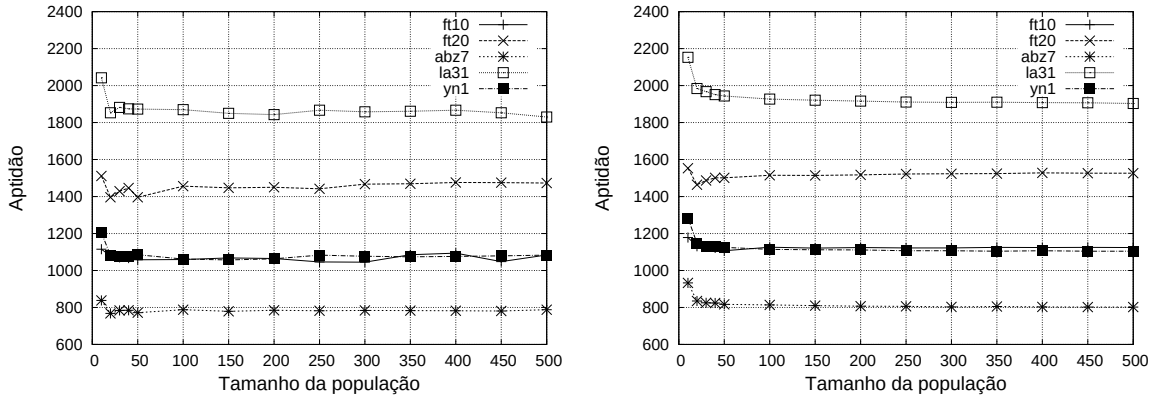
Este capítulo apresenta os resultados obtidos dos experimentos descritos no Capítulo 5, dispostos em gráficos e tabelas. Todos os dados encontrados foram analisados e discutidos.

A Seção 6.2 expõe os resultados dos casos de teste realizados com o SGA, enquanto a Seção 6.3 relata as soluções encontradas pelos experimentos com o modelo em ilha. Já a Seção 6.4 apresenta uma comparação entre versões do SGA e versões do CGPGA.

6.2 Resultados dos Experimentos do SGA

6.2.1 Tamanho da População

As Figuras 6.1 e 6.2 informam os resultados dos testes variação do tamanho da população, conforme descrito na Seção 5.4.3. É possível perceber a partir deles que aumentar o tamanho da população, pelo menos para os problemas testados, praticamente não surtiu diferenças sobre a média da qualidade dos escalonamentos encontrados após 50 indivíduos na população. Não apenas isso, uma população pequena, como a de tamanho 20 para o problema ft20 e tamanho 50 para o problema ft10, conseguiu encontrar melhores resultados, na média.

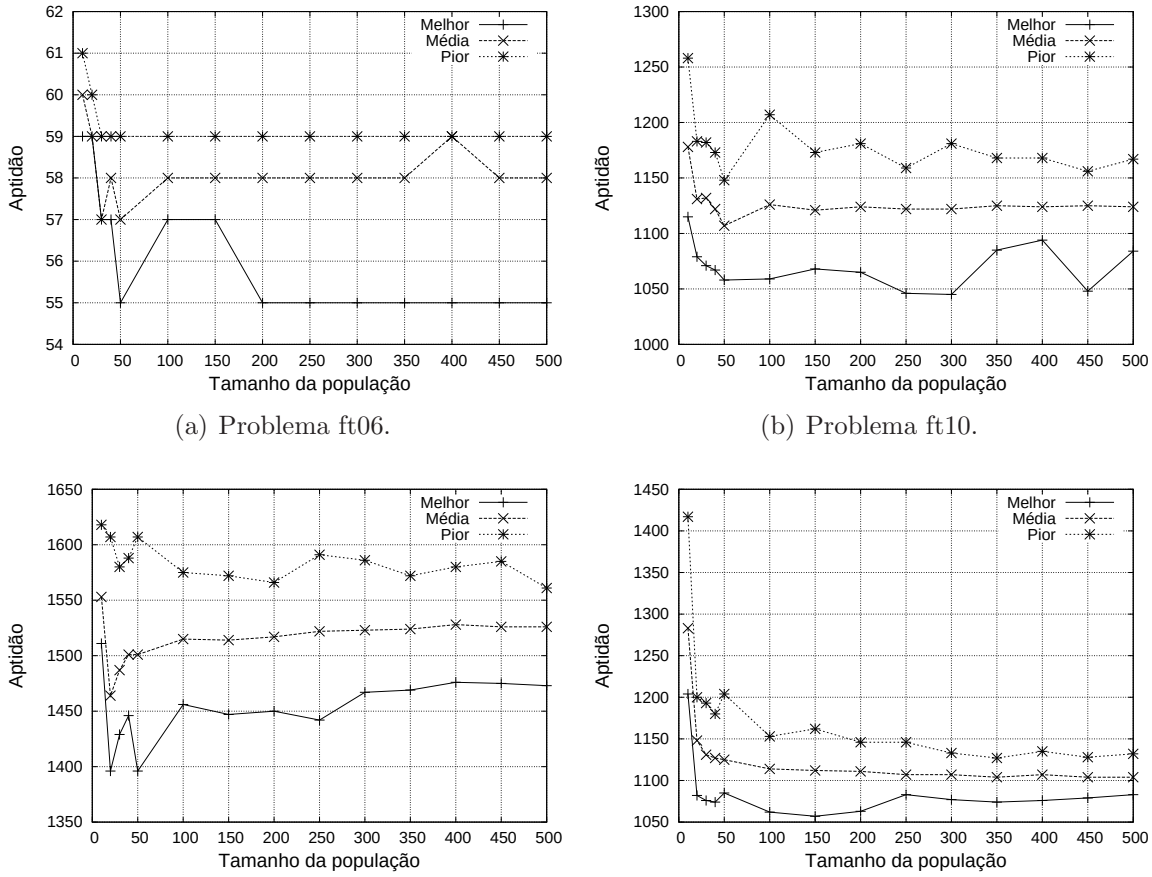


(a) Melhores escalonamentos.

(b) Média dos escalonamentos.

Figura 6.1: Gráfico dos melhores e das médias dos resultados da variação do tamanho da população.

As Figuras 6.3 e 6.4 apresentam os resultados da variação do tamanho da população com elitismo total. O que seria intuitivamente esperado é que o elitismo melhoraria a



(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

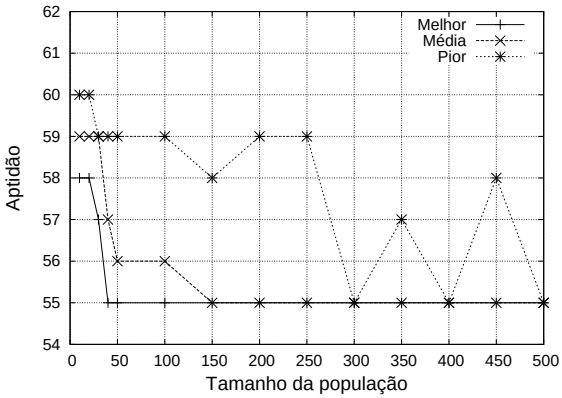
(d) Problema yn1.

Figura 6.2: Gráfico dos resultados da variação do tamanho da população para 4 problemas distintos.

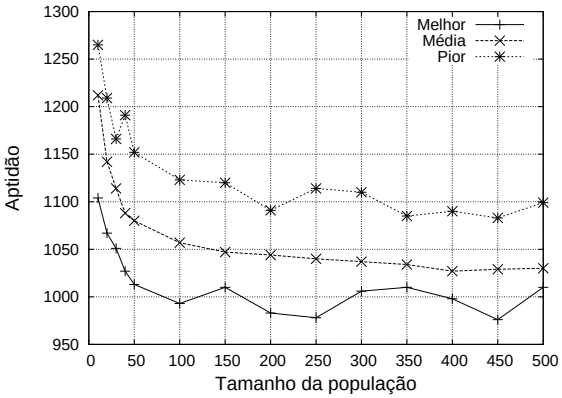
qualidade das soluções no geral, enquanto manteria as mesmas propriedades dos gráficos da variação do tamanho da população sem elitismo (Figuras 6.2 e 6.1). Porém, apenas a primeira constatação é visível (como também mostram os resultados da Tabela 6.6). Para a segunda, os resultados com populações de tamanho pequeno não mais fogem do resto do gráfico, e sim mantêm-se melhor que os anteriores e pior que os sucessores.

6.2.2 Quantidade de Gerações

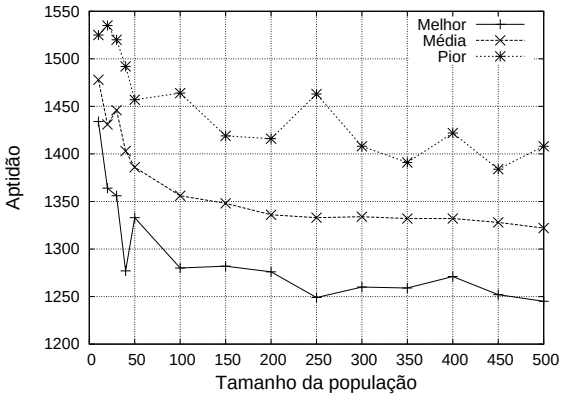
As Figuras 6.5 e 6.6 apresentam os resultados da variação do número de gerações e seu impacto sobre a qualidade dos escalonamentos encontrados. De 10 gerações para 100, houve uma diferença de mais de 200, para melhor, na média dos escalonamentos encontrados para o problema ft10, 100 para o problema ft20. A partir de 50 gerações, o resultado começa a se estabilizar, com uma leve diferença para 100 gerações e diferenças pequenas para quantidades maiores de gerações, proporcionalmente ao tamanho do problema. Isso pode estar relacionado com o fato do algoritmo ter encontrado um mínimo local a partir de um limiar que cresce junto com o tamanho do problema.



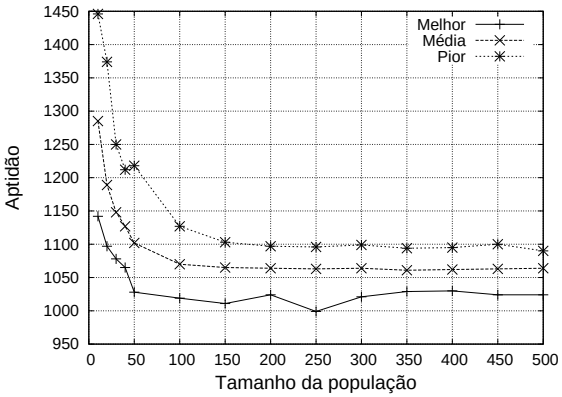
(a) Problema ft06.



(b) Problema ft10.

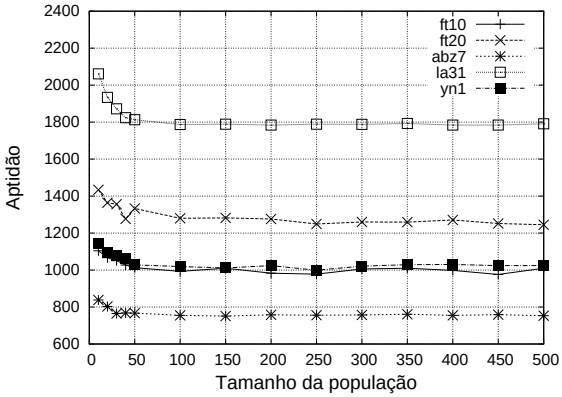


(c) Problema ft20.

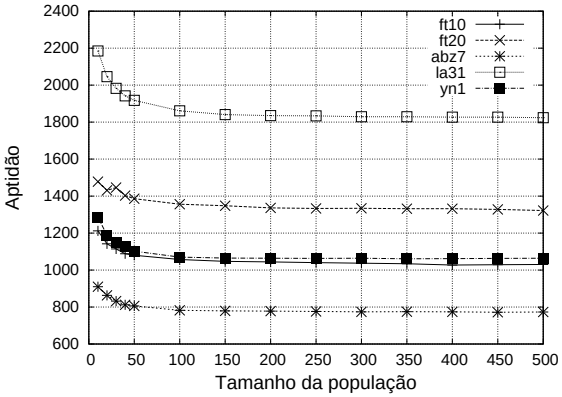


(d) Problema yn1.

Figura 6.3: Gráfico dos resultados da variação do tamanho da população com elitismo total para 4 problemas distintos.



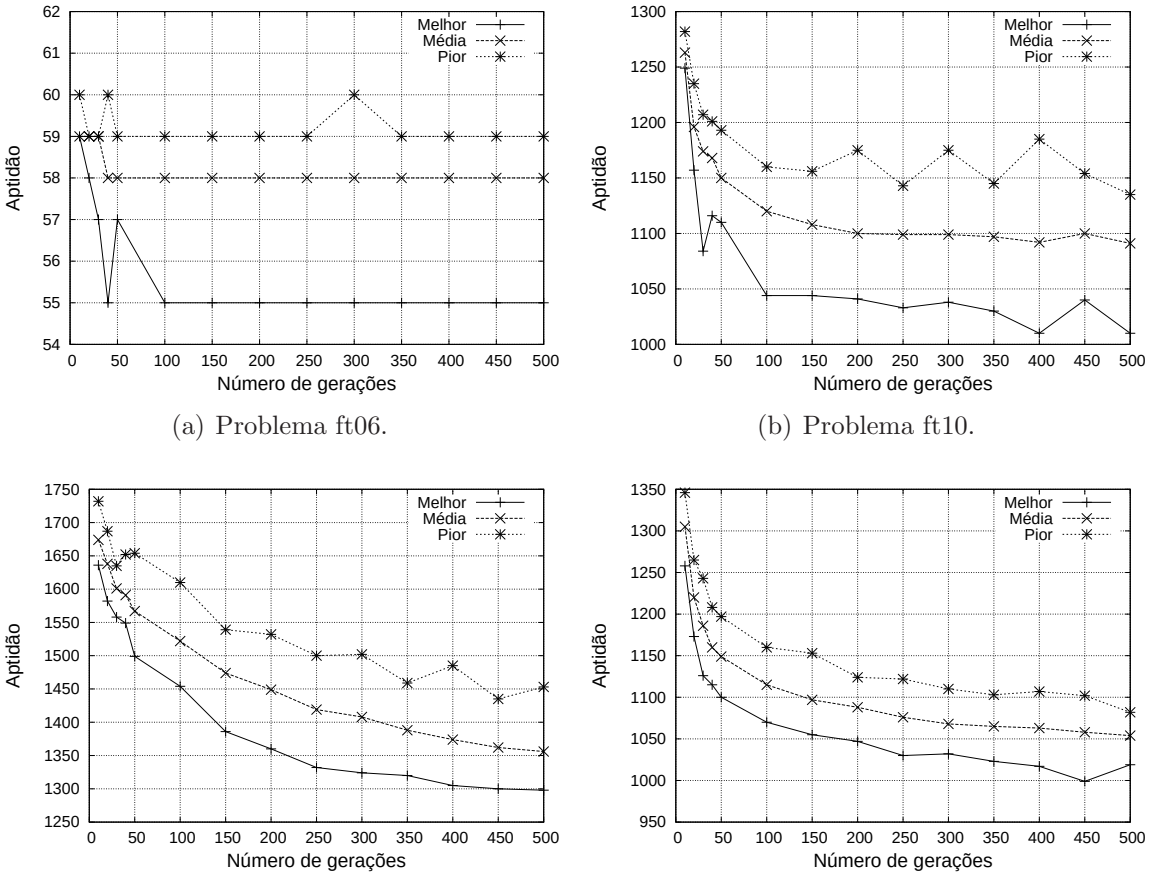
(a) Melhores escalonamentos.



(b) Média dos escalonamentos.

Figura 6.4: Gráfico dos melhores e das médias dos resultados da variação do tamanho da população com elitismo total.

As Figuras 6.7 e 6.8 apresentam os resultados da variação do número de gerações com elitismo total. Com o elitismo total, os melhores indivíduos não são perdidos ao longo das gerações. Como nos resultados obtidos dos experimentos da quantidade de gerações sem



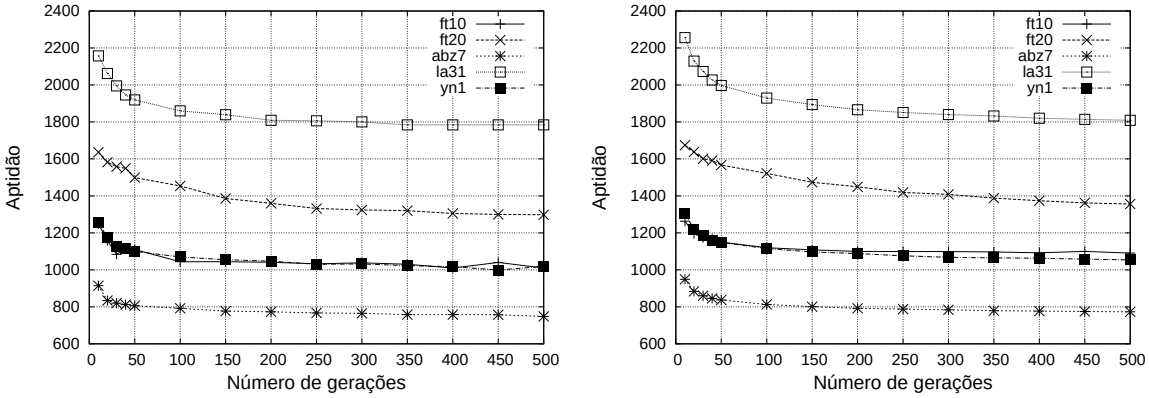
(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

Figura 6.5: Gráfico dos resultados da variação da quantidade de gerações para 4 problemas distintos.



(a) Melhores escalonamentos.

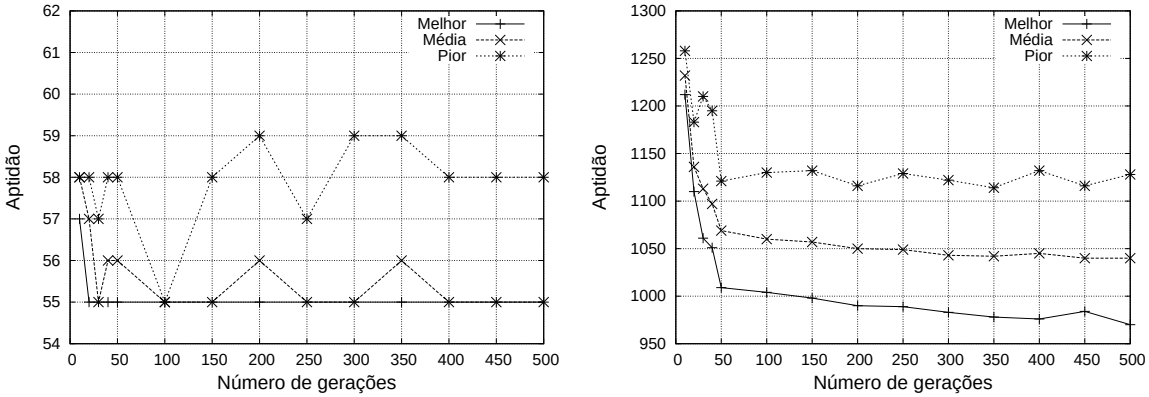
(b) Média dos escalonamentos.

Figura 6.6: Gráfico dos melhores e das médias dos resultados da variação da quantidade de gerações.

elitismo (Figuras 6.5 e 6.6), houve uma estabilização na qualidade dos resultados, apenas com leve melhoras, a partir de 100 gerações. A diferença encontra-se, graças ao elitismo, no aumento aumento considerável da qualidade dos cromossomos durante os experimentos

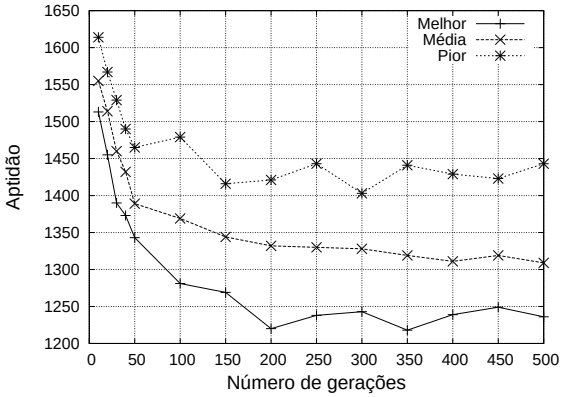
(ou seja, uma aptidão melhor, com um *makespan* menor).

A leve melhora nos resultados ao longo do aumento da quantidade de gerações parece estar associada com o encontro de um mínimo local, do qual o algoritmo tenta se afastar através das perturbações causadas pela mutação. Como a taxa de mutação é pequena, o algoritmo acaba ficando "preso" ao mínimo local. Outro fator que pode indicar melhor esse fato é a diferença entre os cromossomos: é possível que os escalonamentos representados pelos indivíduos (além dos próprios indivíduos) sejam os mesmos ou bem semelhantes.

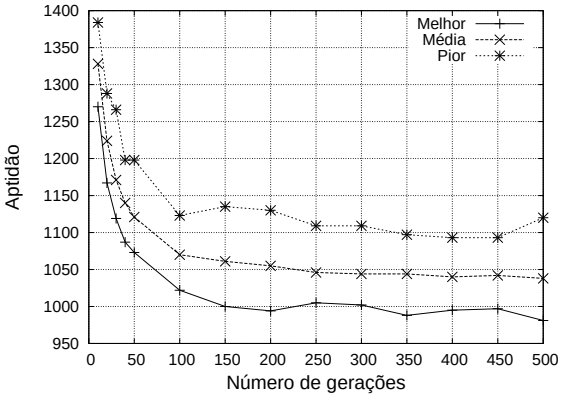


(a) Problema ft06.

(b) Problema ft10.



(c) Problema ft20.

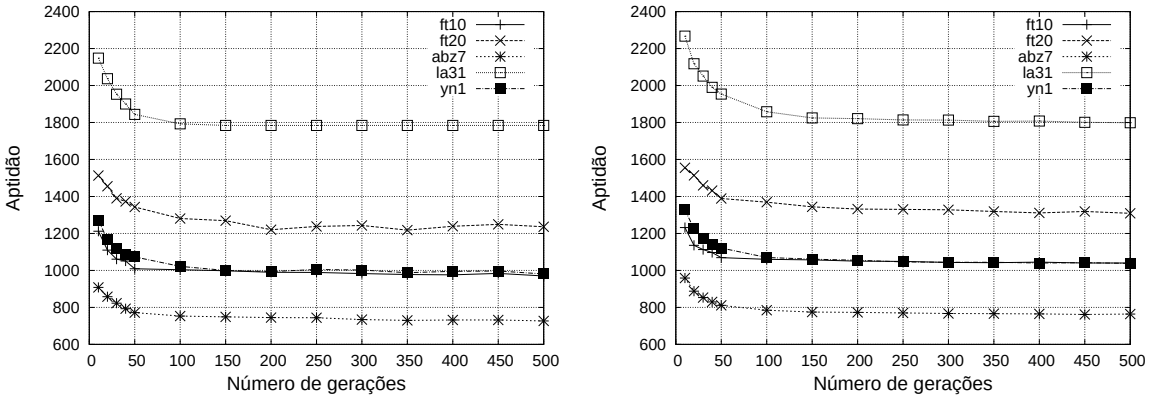


(d) Problema yn1.

Figura 6.7: Gráfico dos resultados da variação da quantidade de gerações com elitismo total para 4 problemas distintos.

6.2.3 Taxas para o Cruzamento e a Mutação

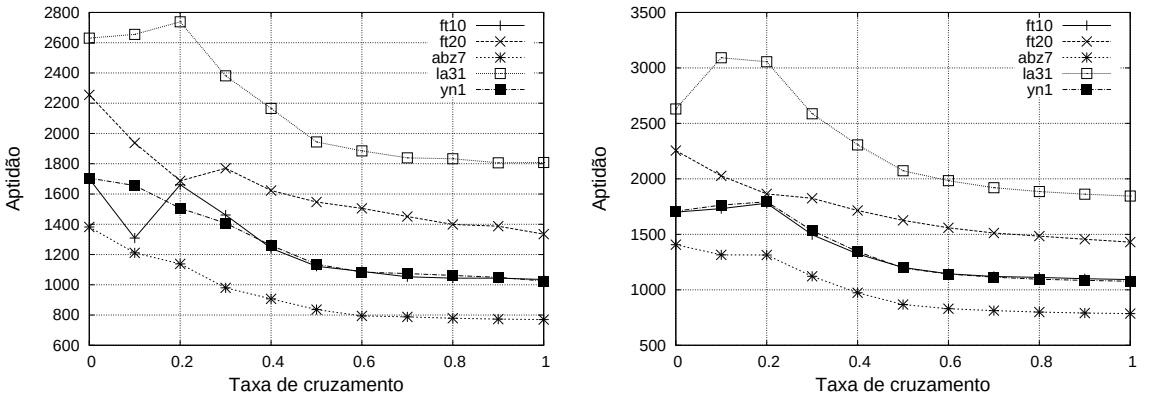
As Figuras 6.9 e 6.10 apresentam os resultados obtidos dos experimentos da variação da taxa de cruzamento. Até 0.4, há algumas pertubações nos resultados, todavia já é possível identificar a tendência de melhora na qualidade dos escalonamentos encontrados. Depois, essa melhora continua até o valor máximo de 1.0. Pode-se perceber também que, em problemas onde a solução ótima é encontrada (como o ft06, cujo escalonamento ótimo possui *makespan* de 55), o aumento na taxa de cruzamento serve apenas para aproximar



(a) Melhores escalonamentos. (b) Média dos escalonamentos.

Figura 6.8: Gráfico dos melhores e das médias dos resultados da variação da quantidade de gerações com elitismo total.

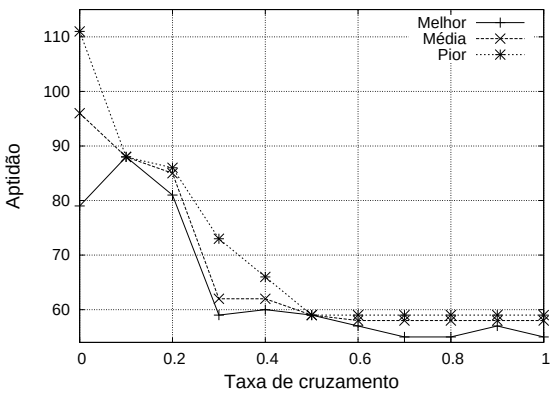
a média e as piores soluções das melhores. Além disso, essa melhora possui uma relação direta com o tamanho do problema, já que à medida que o problema cresce, a melhora entre as taxas também cresce.



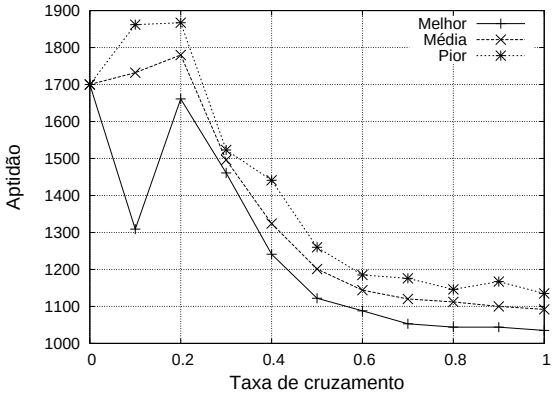
(a) Melhores escalonamentos. (b) Média dos escalonamentos.

Figura 6.9: Gráfico dos melhores e das médias dos resultados da variação da taxa de cruzamento.

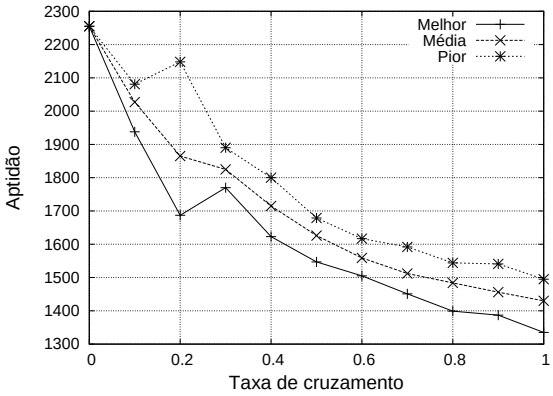
As Figuras 6.11 e 6.12 apresentam os resultados dos testes da variação da taxa de mutação. Assim como o que ocorreu com a os resultados pela variação da taxa de cruzamento, o aumento na taxa de mutação até o máximo de 0.2 demonstrou uma leve melhora, positivamente relativa ao tamanho do problema (quanto maior, melhor). Todavia, essa melhora é extremamente menor quando comparada aos resultados da taxa de cruzamento, confirmando como a operação de mutação é um fator secundário para a qualidade das soluções encontradas.



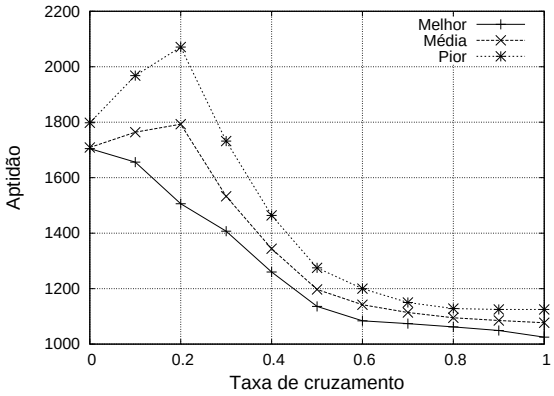
(a) Problema ft06.



(b) Problema ft10.

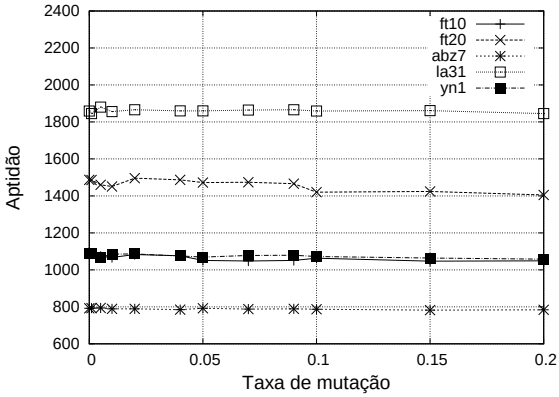


(c) Problema ft20.

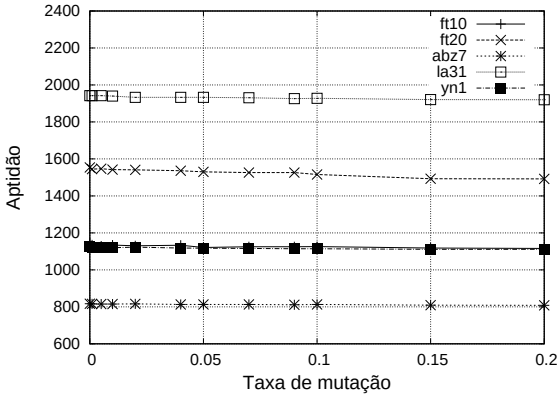


(d) Problema yn1.

Figura 6.10: Gráfico dos resultados da variação da taxa de cruzamento para 4 problemas distintos.

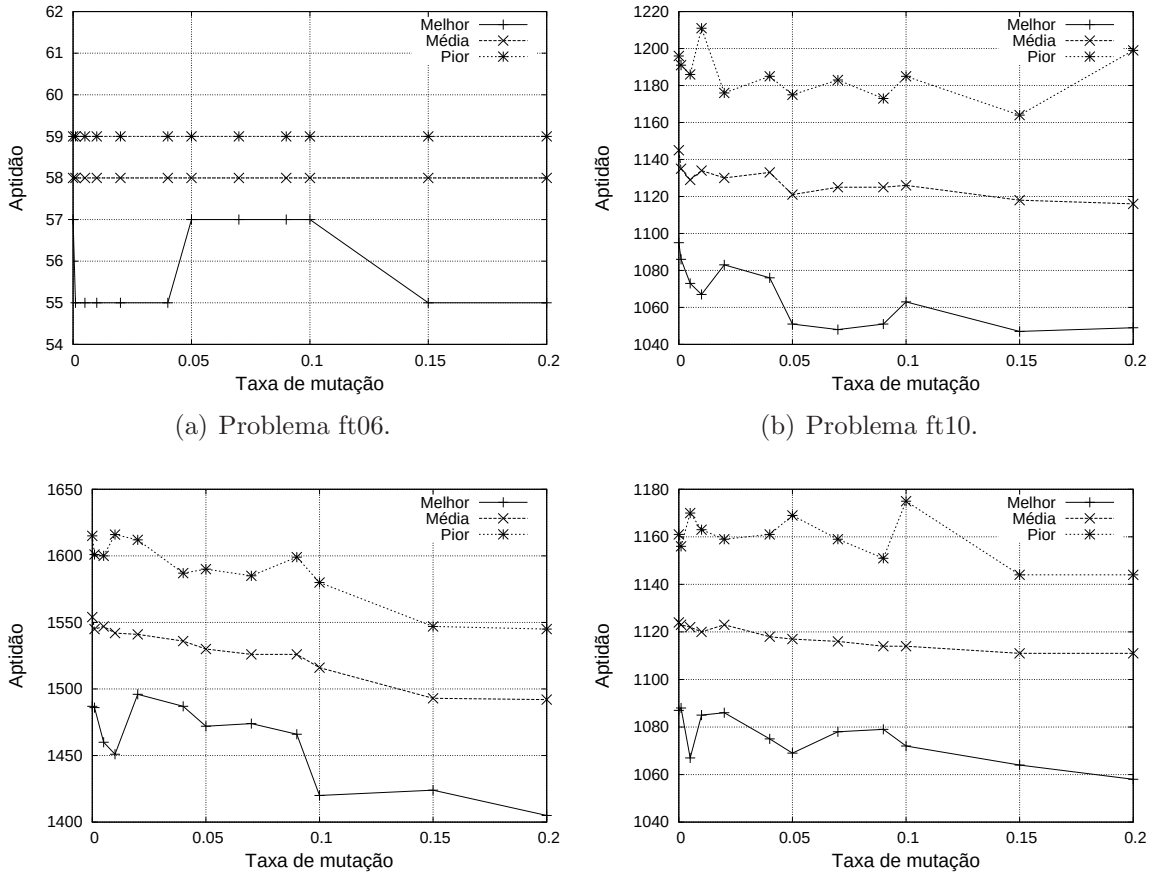


(a) Melhores escalonamentos.



(b) Média dos escalonamentos.

Figura 6.11: Gráfico dos melhores e das médias dos resultados da variação da taxa de mutação.



(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

Figura 6.12: Gráfico dos resultados da variação da taxa de mutação para 4 problemas distintos.

6.2.4 Alternativas para os Operadores

A Tabela 6.1 mostra os resultados para os experimentos do algoritmo utilizado para a geração da população inicial de indivíduos. Ela mostra como o algoritmo de G&T atinge as melhores médias em todos os seis problemas e, com exceção do ft06, também possibilitou encontrar os melhores resultados, quando comparado com o algoritmo de geração aleatória de cromossomos.

Problema	Referência	Aleatório		G&T	
		Melhor	Média	Melhor	Média
ft06	55	55	**58**	58	**58**
ft10	930	1055	1116	1053	**1084**
ft20	1165	1454	1515	1425	**1478**
abz7	685	788	812	770	**795**
la31	1784	1854	1926	1833	**1899**
yn1	925	1067	1115	1037	**1065**

Tabela 6.1: Resultado dos experimentos das alternativas para a geração da população inicial de cromossomos.

A Tabela 6.2 apresenta os melhores e as médias dos resultados encontrados por cada

um dos quatro operadores de cruzamento implementados. Os valores dessa tabela indicam como o operador GPMX consistentemente encontrou melhores resultados para cada um dos seis problemas testados, tanto na média quanto nas melhores soluções encontradas entre as 100 execuções de cada caso de teste. Em segundo lugar, próximo do primeiro, encontra-se o algoritmo PPXm. Já com resultados um pouco mais distantes, encontram-se o "*crossover* 4" em terceiro e o GTF em quarto.

Problema	PPXm		GPMX		"*Crossover* 4"		GTF	
	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	57	59	55	**58**	57	**58**	60	63
ft10	1044	1119	1041	**1107**	1085	1165	1205	1285
ft20	1413	1516	1409	**1482**	1514	1579	1646	1792
abz7	792	814	762	**794**	963	1030	969	1039
la31	1860	1927	1789	**1872**	2175	2324	2301	2511
yn1	1071	1114	1030	**1102**	1372	1502	1311	1459

Tabela 6.2: Resultado dos experimentos das alternativas para a operação de cruzamento.

A Tabela 6.3 agrega os resultados dos experimentos com as seis operações de mutação implementadas. Em primeiro lugar para todos os problemas, exceto o ft10, está o algoritmo de inserção. Para o problema f10, o algoritmo que obteve os melhores resultados foi o de inversão. Os resultados então variam para o segundo lugar de acordo com o problema.

As Tabelas 6.4 e 6.5 contêm os resultados da qualidade do melhor indivíduo entre 100 execuções do programa e da média do melhor escalonamentos encontrado por cada uma das execuções, comparando as operações de seleção testadas. O torneio de 3 se sobressaiu, alcançando as melhores médias em quase todos os problemas (apenas o ft06 que foi, novamente, a exceção). Próximo está a seleção aleatória por classificação, que ficou em segundo lugar nas médias, porém chegou a alcançar alguns dos melhores resultados para os problemas ft06, ft20, la31 e yn1, enquanto a seleção por torneio de 3 alcançou os melhores resultados nos problemas ft06, ft10, abz7 e la31. Ambos alcançaram a solução ótima para o problema la31. Em terceiro lugar encontra-se a seleção por torneio de 2, com as terceiras melhores médias para os problemas abz7, la31 e yn1, além da mesma média para o problema ft06 que as outras duas soluções encontraram. Logo mais estão a classificação linear, que possui as mesmas médias que o torneio de 2 para os problemas abz7 e la31, além da melhor média para o ft06; e o método agressivo, com a mesma e melhor média para o ft06, além das terceiras melhores soluções para o problema ft10 e ft20.

A Tabela 6.6 resume os resultados dos testes com os diferentes tipos de elitismo. Ela demonstra como a abordagem elitista permite uma melhora nos resultados e, por mais que não seja algo existente na natureza, representa um importante mecanismo para a otimização dos algoritmos genéticos.

Problema	Troca		Troca em multiponto		Inversão		Inserção		Deslocamento		Vizinhança	
	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	57	**58**	57	**58**	55	**58**	57	**58**	55	59	55	**58**
ft10	1063	1121	1065	1134	1048	**1117**	1066	1126	1065	1126	1085	1129
ft20	1424	1515	1465	1531	1467	1530	1437	**1512**	1474	1533	1440	1525
abz7	792	812	792	815	786	814	765	**809**	792	815	790	813
la31	1868	1927	1866	1936	1880	1934	1845	**1917**	1860	1939	1854	1930
yn1	1072	1115	1076	1120	1072	1120	1068	**1113**	1082	1118	1077	1121

Tabela 6.3: Resultado dos experimentos das alternativas para a operação de mutação.

Problema	SUS		SUS quadrada		Mais apto		Torneio de 2		Torneio de 3		Torneio de 2, 0.75	
	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	55	58	57	58	55	58	57	58	57	58	55	58
ft10	1045	1120	1044	1125	1054	1118	1032	1094	1012	**1072**	1026	1099
ft20	1448	1512	1447	1506	1268	1391	1327	1396	1305	**1363**	1342	1433
abz7	790	812	788	812	804	861	755	782	735	**772**	759	788
la31	1881	1929	1852	1925	1895	2028	1786	1840	1784	**1808**	1803	1855
yn1	1071	1115	1075	1114	1133	1210	1039	1071	1008	**1055**	1030	1083

Tabela 6.4: Resultado dos experimentos das alternativas para a operação de seleção.

Problema	Agressivo		Classificação linear		Aleatório		Aleatório por clas.		RR	
	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	55	**57**	55	**57**	57	59	55	58	55	58
ft10	1019	1088	1041	1095	1068	1133	1018	1082	1032	1098
ft20	1297	1391	1344	1410	1470	1538	1298	1382	1384	1438
abz7	774	834	749	782	790	821	747	775	779	799
la31	1847	1976	1789	1840	1861	1950	1784	1823	1794	1885
yn1	1071	1143	1024	1074	1090	1131	1004	1060	1042	1094

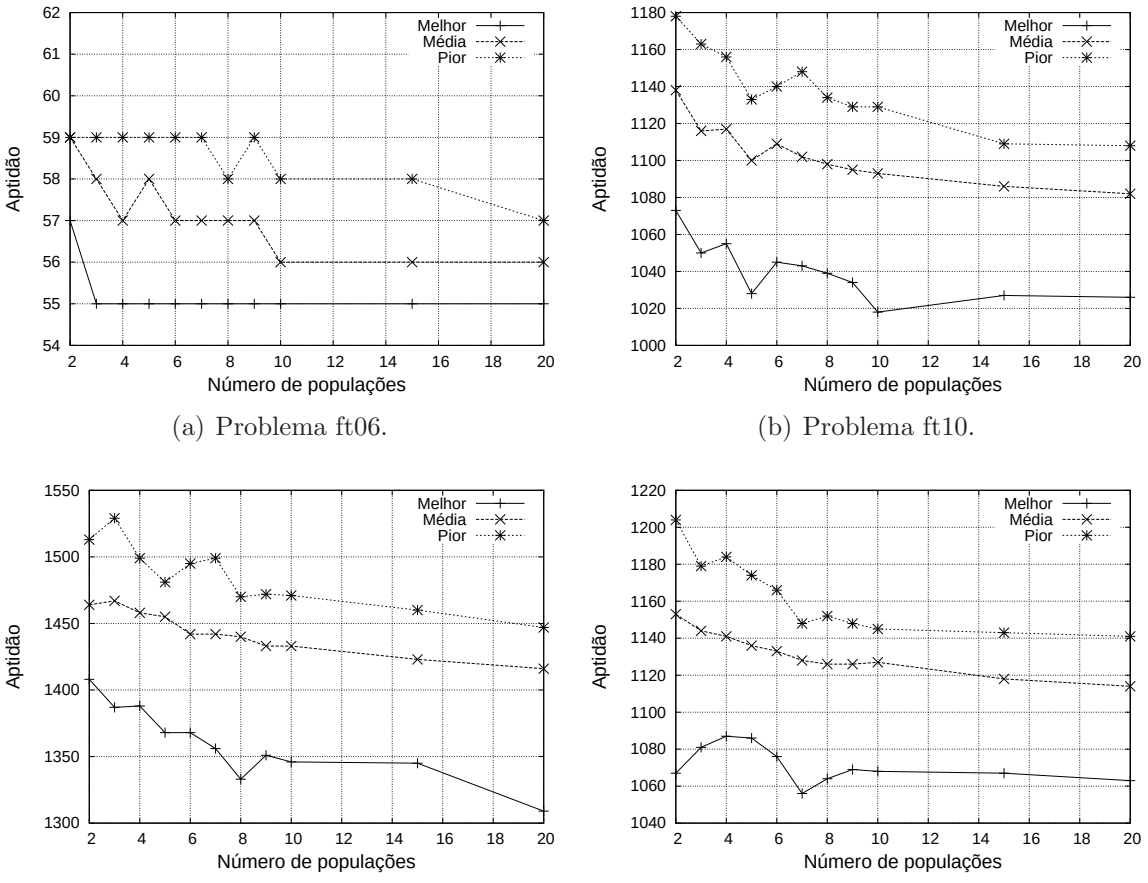
Tabela 6.5: Resultado dos experimentos das alternativas para a operação de seleção. Continuação.

Problema	Nenhum		Mais apto		Total	
	Melhor	Média	Melhor	Média	Melhor	Média
ft06	55	58	55	**56**	55	**56**
ft10	1087	1176	1023	1095	1001	**1061**
ft20	1517	1588	1413	1479	1263	**1343**
abz7	974	1033	891	943	796	**857**
la31	2186	2333	2064	2159	1905	**1998**
yn1	1395	1516	1294	1387	1137	**1227**

Tabela 6.6: Resultado dos experimentos das formas de elitismo.

6.3 Resultados dos Experimentos com o Modelo em Ilha

As Figuras 6.13 e 6.14 mostram os resultados dos testes da variação do número de populações no modelo em ilha. À medida que o número de populações aumenta, também aumenta a qualidade das soluções encontradas. Contudo, essa melhora é extremamente menor que a proporção do aumento do tamanho das populações. Por exemplo, a média do *makespan* do problema yn1 diminuiu de 1153 com 2 ilhas para 1114 com 20 ilhas.



(a) Problema ft06.

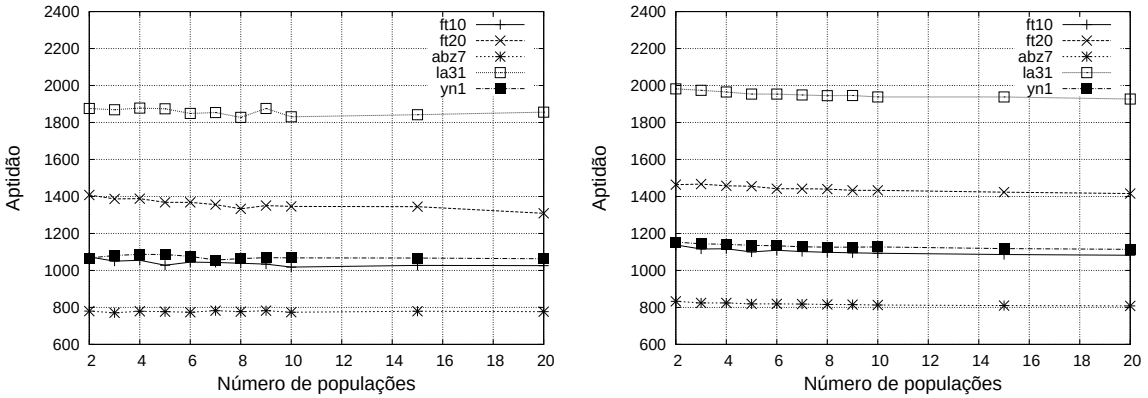
(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

Figura 6.13: Gráfico dos resultados da variação do número de populações no modelo em ilha para 4 problemas distintos.

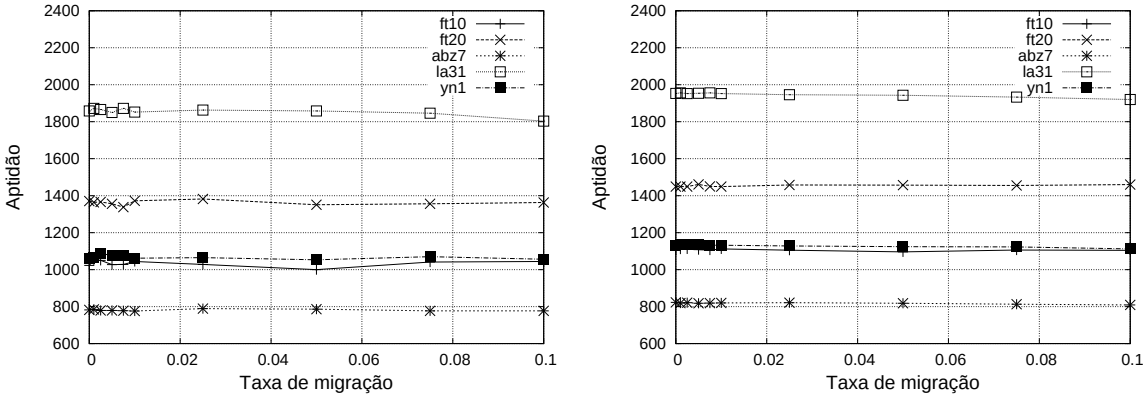
As Figuras 6.15 e 6.16 apresentam os resultados dos experimentos da variação da taxa



(a) Melhores escalonamentos. (b) Média dos escalonamentos.

Figura 6.14: Gráfico dos melhores e das médias dos resultados da variação da quantidade de populações no modelo em ilha.

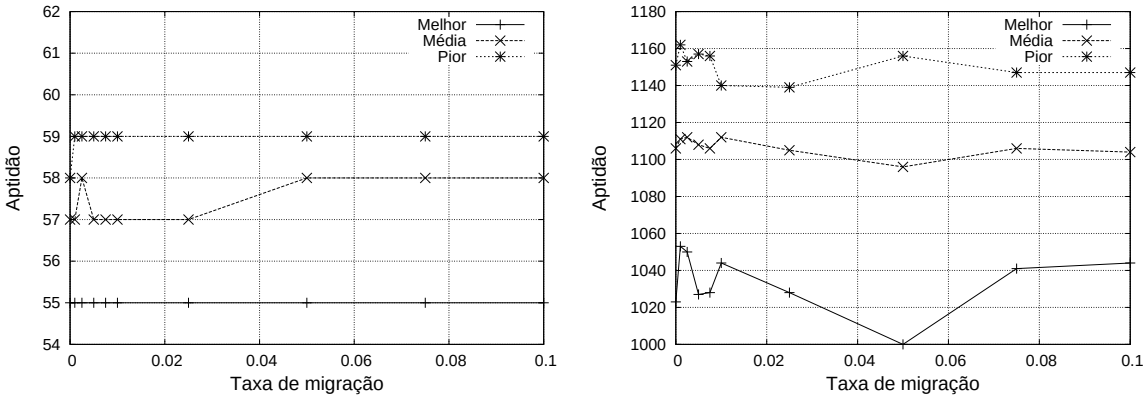
de migração. Nota-se que o aumento dessa taxa manteve ou tornou piores as soluções encontradas pelos problemas ft06, ft10 e ft20, com exceção do valor 0.05 para o ft10, que obteve uma média menor, inclusive a melhor entre as soluções com outras taxas. Já para os problemas abz7, la31 e yn1, as soluções se mantiveram no início e melhoraram a partir de um momento. Isso indica como o tamanho do problema está diretamente relacionado com a taxa a ser utilizada. Outro fator importante para se notar é o tamanho pequeno das *demes*. Talvez com uma subpopulação maior essas diferenças entre os tamanhos dos problemas tornem-se mais visíveis.



(a) Melhores escalonamentos. (b) Média dos escalonamentos.

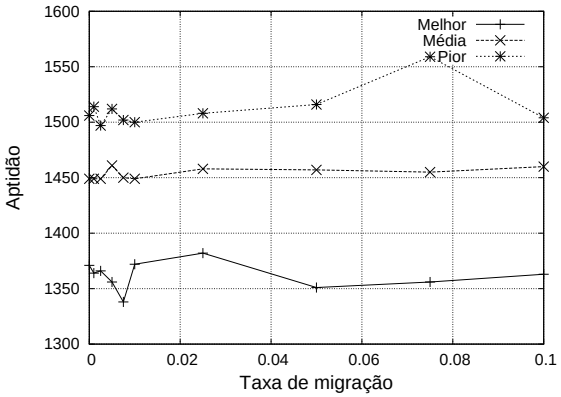
Figura 6.15: Gráfico dos melhores e das médias dos resultados da variação da taxa de migração.

As Figuras 6.17 e 6.18 apresentam os resultados dos experimentos da variação da taxa de migração com *demes* de tamanho 100. Neles, a variação da taxa manteve ou piorou todos os resultados, todavia de forma mínima, dos problemas ft06, ft10 e ft20, enquanto melhorou, também minimamente, as soluções encontradas para os problemas abz7, la31

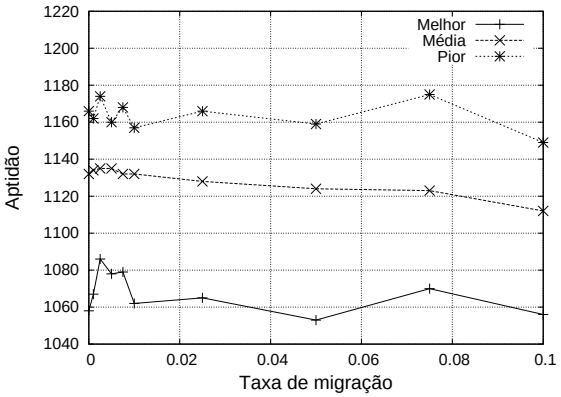


(a) Problema ft06.

(b) Problema ft10.



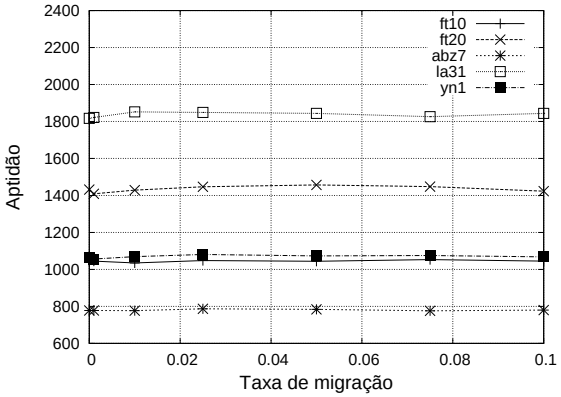
(c) Problema ft20.



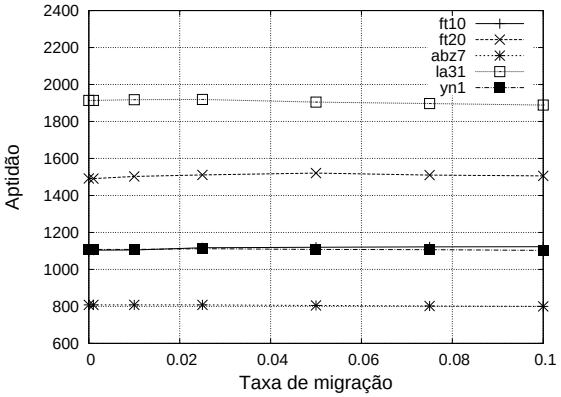
(d) Problema yn1.

Figura 6.16: Gráfico dos resultados da variação da taxa de migração para 4 problemas distintos.

e yn1, após pioras iniciais.

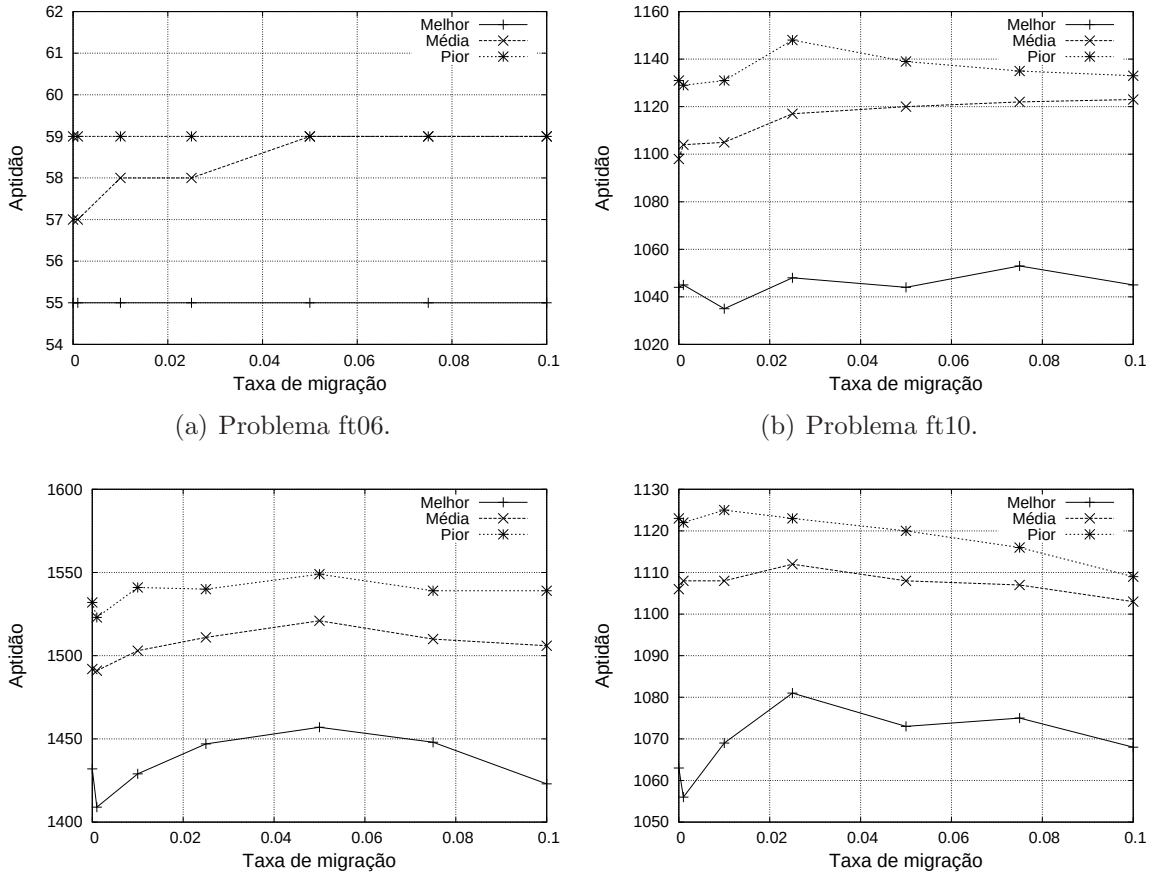


(a) Melhores escalonamentos.



(b) Média dos escalonamentos.

Figura 6.17: Gráfico dos melhores e das médias dos resultados da variação da taxa de migração com subpopulações de tamanho 100.



(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

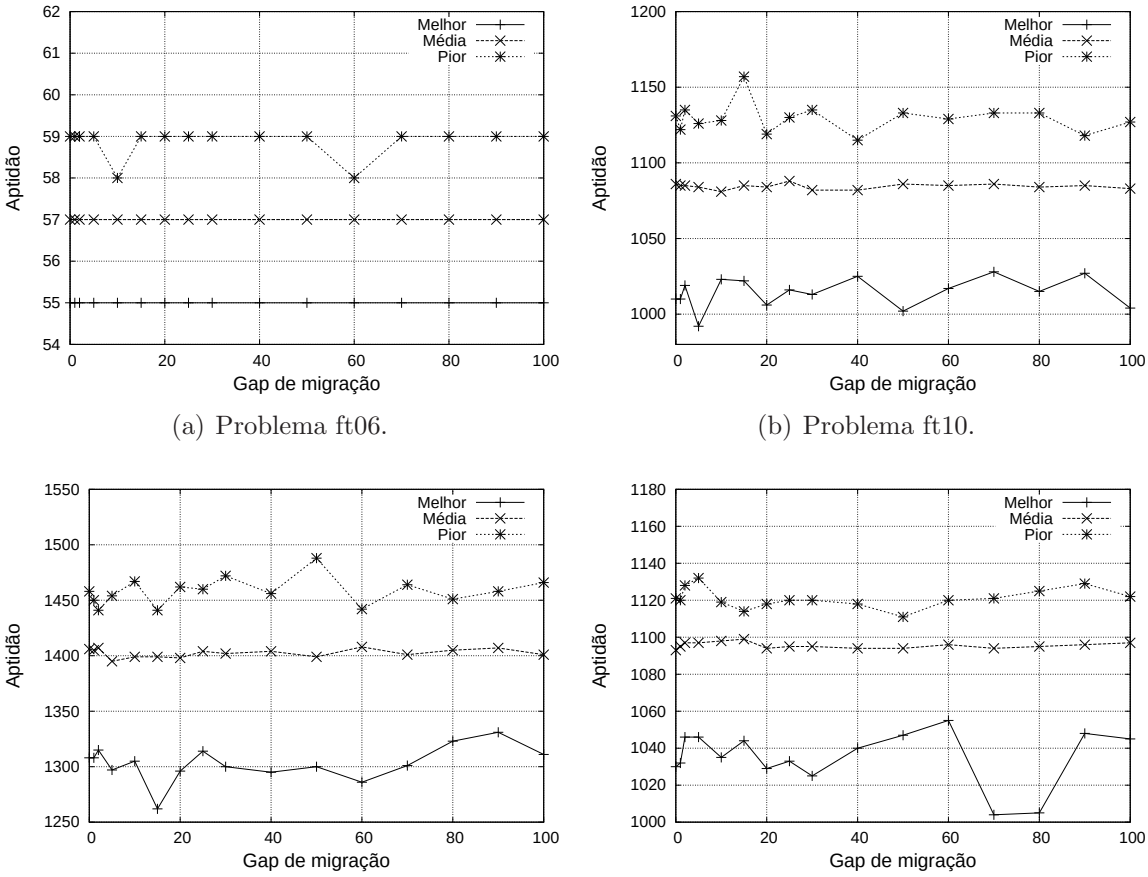
Figura 6.18: Gráfico dos resultados da variação da taxa de migração com, subpopulações de tamanho 100, para 4 problemas distintos.

6.3.1 Intervalo de Migração

As Figuras 6.19 e 6.20 exibem os resultados encontrados para o caso de teste do intervalo de migração com taxa de 0.001. Percebe-se que a variação do *gap* de migração praticamente não surtiu efeito sobre os resultados em uma configuração de subpopulações de tamanho 20 e 500 gerações. Talvez aumentando o tamanho da taxa de migração ou das *demes* os resultados variem.

As Figuras 6.21 e 6.22 são os resultados do quinto caso de teste descrito na Seção 5.5. Ele complementa o caso de teste anterior, aumentando a taxa de migração para 0.1. Desta vez, houve uma melhora para todos os problemas, exceto o ft06, quando o intervalo é a cada 1 geração (em toda geração a migração ocorre).

As Figuras 6.23 e 6.24 expõem os resultados do sexto caso de teste relatado na Seção 5.5, que complementa os últimos dois experimentos aumentando o tamanho das subpopulações para 100 indivíduos cada e a taxa de migração modificada para 0.05. Nesta conjectura, a variação do intervalo de migração praticamente não causou diferenças na qualidade das soluções encontradas.



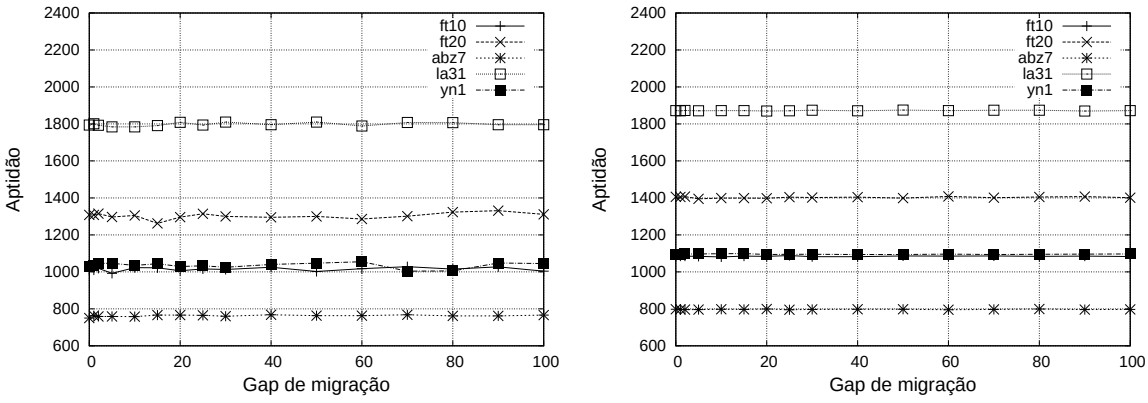
(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

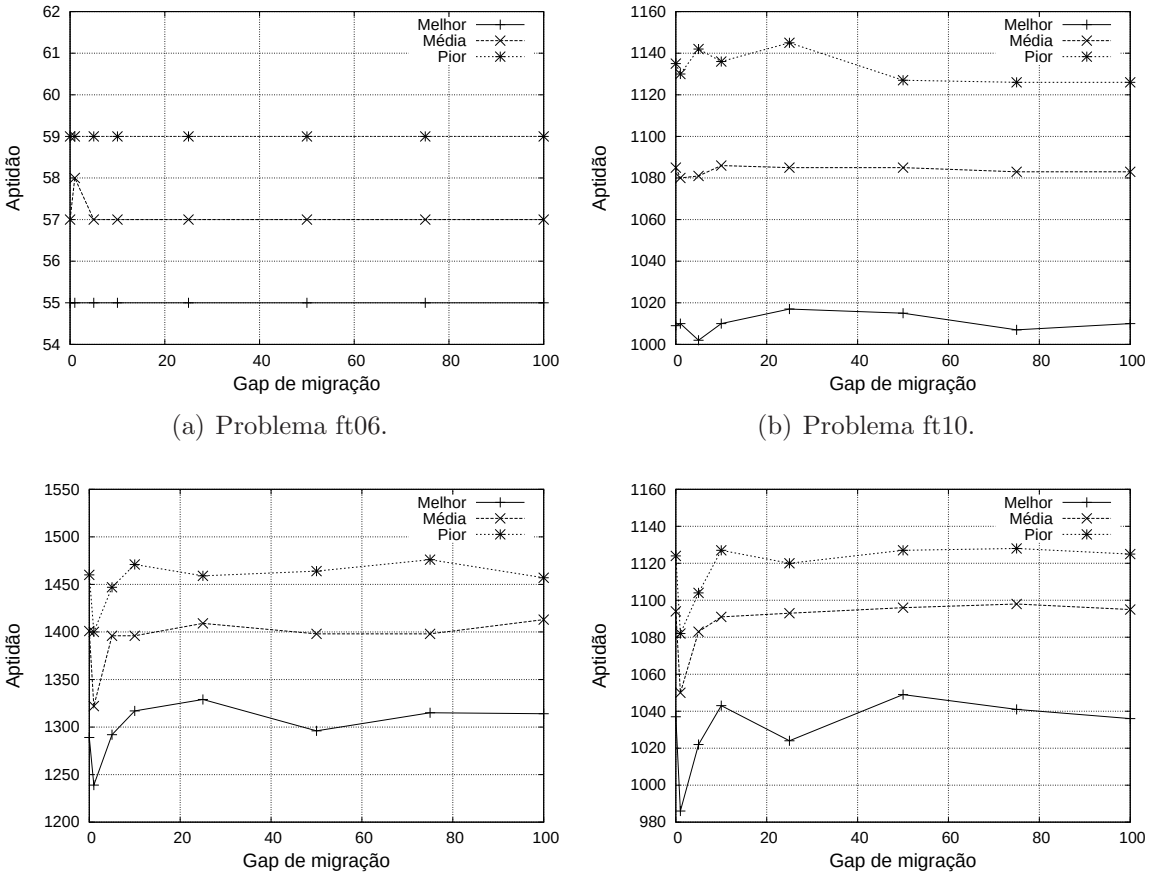
Figura 6.19: Gráfico dos resultados da variação do intervalo de migração para 4 problemas distintos, com taxa de migração igual a 0.001.



(a) Melhores escalonamentos.

(b) Média dos escalonamentos.

Figura 6.20: Gráfico dos melhores e das médias dos resultados da variação do intervalo de migração, com taxa igual a 0.001.



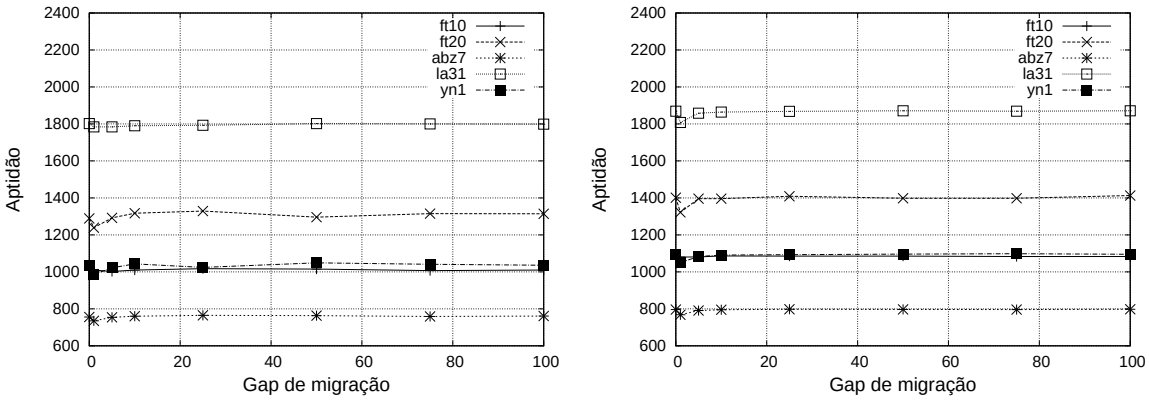
(a) Problema ft06.

(b) Problema ft10.

(c) Problema ft20.

(d) Problema yn1.

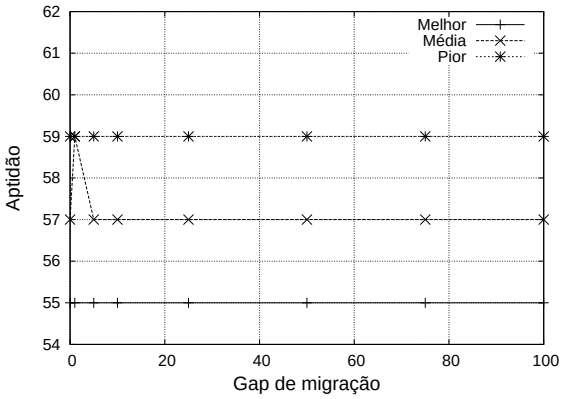
Figura 6.21: Gráfico dos resultados da variação do intervalo de migração para 4 problemas distintos, com taxa de migração igual a 0.1.



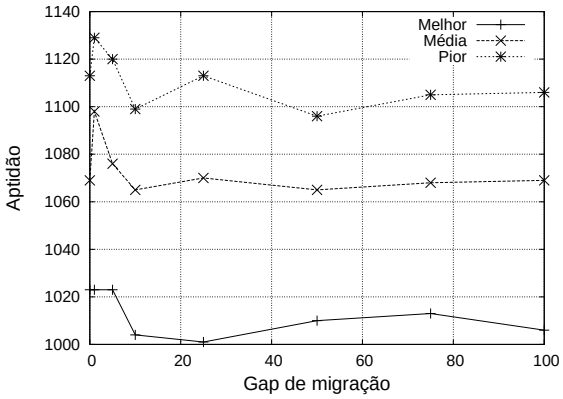
(a) Melhores escalonamentos.

(b) Média dos escalonamentos.

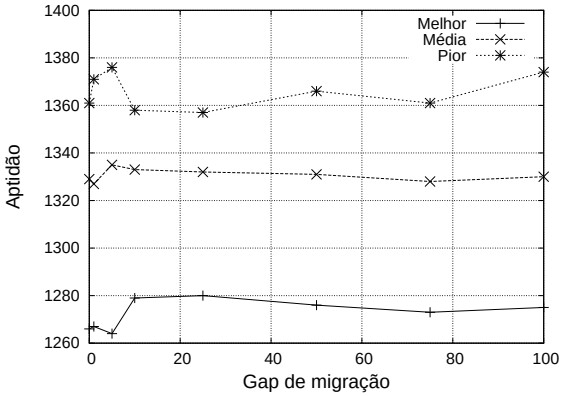
Figura 6.22: Gráfico dos melhores e das médias dos resultados da variação do intervalo de migração, com taxa igual a 0.1.



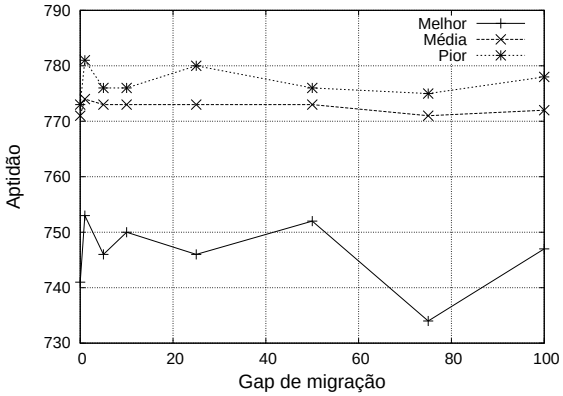
(a) Problema ft06.



(b) Problema ft10.

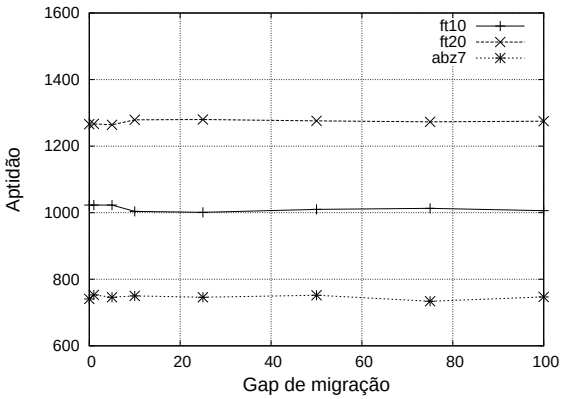


(c) Problema ft20.

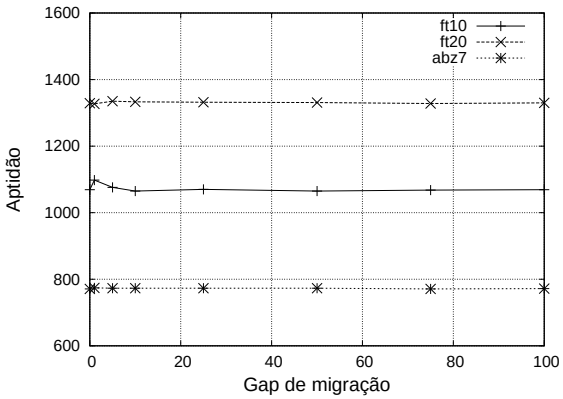


(d) Problema abz7.

Figura 6.23: Gráfico dos resultados da variação do intervalo de migração para 4 problemas distintos, com taxa igual a 0.05.



(a) Melhores escalonamentos.



(b) Média dos escalonamentos.

Figura 6.24: Gráfico dos melhores e das médias dos resultados da variação do intervalo de migração, com taxa fixada em 0.05.

6.4 Resultados dos Experimentos Comparativos entre SGA e DGA

As Tabelas 6.7 e 6.8 reúnem as soluções encontradas, respectivamente, pelas configurações do SGA e do DGA, conforme detalhado na Seção 5.6. A versão SGA1 apresenta os piores resultados, conforme esperado. O SGA2, com melhores parâmetros, apresenta uma melhora apenas quanto ao SGA1, ficando em terceiro lugar. A combinação SGA4, ao contrário do esperado, fica em segundo lugar. Isso indica que cada combinação de operadores possui uma configuração ótima dos parâmetros (tamanho da *deme*, quantidade de gerações e taxas de cruzamento e mutação), que pode ou não ser semelhante à de outros operadores. Assim, o SGA3, que com os melhores operadores, fica em primeiro lugar na qualidade das soluções (média e melhores) encontradas pelo modelo tradicional do GA.

Quanto aos testes com o modelo em ilha, os melhores resultados são obtidos pela combinação DGA4 dos melhores operadores com os melhores parâmetros (segundo os testes apresentados nas seções anteriores da versão SGA), com a única exceção sendo o problema ft06. Isso significa que o modelo modificou a relação entre os operadores e os parâmetros usados pois, ao contrário dos resultados do SGA4, o DGA4 encontrou soluções de melhor qualidade do que apenas o uso dos melhores operadores no DGA3, que ficou em terceiro lugar, ou dos melhores parâmetros no DGA2, que ficou em segundo lugar. Aliás, no modelo em ilha, o uso dos melhores parâmetros surtiu um efeito maior do que os operadores quanto à maior qualidade dos escalonamentos encontrados, ao contrário do ocorrido com o SGA. Entretanto, o uso de maiores taxas (de cruzamento e mutação) e maior quantidade de gerações implicou em um grande aumento no tempo das 100 execuções independentes: enquanto o DGA3 demorou em torno de 10 minutos, o DGA2 demorou 4 horas e o DGA4 levou 5 horas e 8 minutos para finalizar.

Em quarto lugar ficou o DGA5, usando heterogeneidade. Essa versão, por mais que tenha encontrado melhores escalonamentos apenas do que a homogênea DGA1, parece ser promissora quando bem planejada, especialmente ao usar operadores que "sigam" caminhos bem distintos para encontrar soluções de maior qualidade. No entanto, sua complexidade é maior, sendo difícil prever se uma dada configuração será melhor do que outra.

Em último lugar encontra-se o DGA1. Este superou o SGA1 em tudo nos problemas ft06, ft10 e ft20, no quesito melhor escalonamento nos problema abz7 e la31, obtendo piores médias, enquanto foi pior em tudo no problema yn1.

A Tabela 6.9 compara a melhor configuração do SGA com a melhor do modelo em ilha. Percebe-se como o DGA4 supera completamente o SGA3. Ambos encontraram a solução ótima para os problemas ft06 e la31, porém a média do DGA4 em todos os problemas é bem menor, com quase o mesmo valor dos melhores resultados encontrados pelo SGA3 nos problemas ft20 e yn1.

Por fim, os resultados dos experimentos realizados indicam como o modelo em ilha consegue superar o modelo SGA no geral, mas necessita de uma devida atenção aos parâmetros e operadores usados, bem como a qual problema o algoritmo está sendo aplicado, pois é possível que a versão do DGA seja pior do que o modelo tradicional.

Problema	Referência	SGA1		SGA2		SGA3		SGA4	
		Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	55	55	**58**	55	**58**	56	**58**	58	59
ft10	930	1044	1121	1010	1092	995	**1043**	994	1060
ft20	1165	1455	1515	1268	1385	1257	**1319**	1242	1326
abz7	685	785	811	755	792	728	**759**	734	766
la31	1784	1864	1923	1784	1863	1784	**1828**	1784	1841
yn1	925	1082	1116	1033	1094	998	**1033**	989	1037

Tabela 6.7: Resultado dos experimentos das diferentes configurações do SGA.

Problema	Referência	DGA1		DGA2		DGA3		DGA4		DGA5	
		Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média	Melhor	Média
ft06	55	55	**56**	55	**56**	55	58	55	57	55	57
ft10	930	1024	1101	976	1028	1024	1073	942	**1017**	975	1036
ft20	1165	1392	1459	1223	1299	1297	1356	1180	**1258**	1219	1297
abz7	685	784	821	729	758	739	771	716	**744**	725	754
la31	1784	1848	1959	1784	1802	1784	1860	1784	**1805**	1784	1815
yn1	925	1093	1134	983	1029	989	1048	958	**999**	971	1018

Tabela 6.8: Resultado dos experimentos das diferentes configurações do DGA.

Problema	Referência	SGA3		DGA4	
		Melhor	Média	Melhor	Média
ft06	55	56	58	55	**57**
ft10	930	995	1043	942	**1017**
ft20	1165	1257	1319	1180	**1258**
abz7	685	728	759	716	**744**
la31	1784	1784	1828	1784	**1805**
yn1	925	998	1033	958	**999**

Tabela 6.9: Comparação entre a melhor configuração do SGA com a melhor do DGA.

CAPÍTULO 7

CONCLUSÃO

Este trabalho apresentou os fundamentos básicos sobre os Algoritmos Genéticos, seus modelos paralelos e sobre o problema de *Job Shop Scheduling*, possibilitando compreender a relação entre eles e como as heurísticas baseadas no princípio da evolução podem ser usadas para solucionar esse problema de otimização.

Com a definição desses fundamentos, diversos parâmetros e operadores, bem como os modelos tradicionais e em ilha, foram estudados, implementados e seus resultados analisados, buscando as configurações mais adequadas para que os GAs encontrem as melhores soluções para o problema de JSS. Os seguintes parâmetros foram testados: o tamanho da população, o número de gerações, a taxa de cruzamento, a taxa de mutação, a quantidade de populações, a taxa de migração e o intervalo de migração. Estes últimos três itens estão relacionados com o modelo em ilha. Os seguintes operadores foram experimentados: a geração da população inicial, a seleção, o cruzamento, a mutação e a estratégia elitista. Para o modelo DGA, a topologia de comunicação usada foi a de um anel unidirecional.

Os resultados indicam que uma população menor pode às vezes encontrar soluções melhores e que há uma relação entre o tamanho da população e a estratégia elitista, pois com elitismo o aumento no tamanho da população seguiu melhorando a qualidade dos escalonamentos encontrados até certo tamanho, a partir do qual a solução fica praticamente estagnada. Essa parada no melhoramento da aptidão também ocorre com o número de gerações, sendo mais cedo (porém em um valor melhor) quando o elitismo "total" foi usado.

Os resultados da taxa de cruzamento indicam que o aumento da taxa tende a melhorar a qualidade das soluções encontradas pelo algoritmo, mas que pode haver exceções em alguns valores, dependendo do problema e outras configurações, como o uso ou não de uma estratégia elitista. Com a taxa de mutação, os valores oscilam, com uma leve e não tão constante melhora no aumento da taxa.

Os melhores operadores, segundo os testes feitos, são: G&T adaptado, para a geração da população inicial; Torneio de 3 para a seleção; GPMX para o cruzamento; inserção para a mutação; e elitismo "total" entre as abordagens elitistas.

Para o modelo em ilha, tanto baixas taxas de migração quanto valores pequenos para o intervalo de migração têm efeitos variados sobre a qualidade dos escalonamentos encontrados. À medida que esses valores aumentam, há uma piora ou uma estagnação do *fitness* das médias das execuções.

Finalmente, os experimentos comparativos entre o SGA e o DGA indicam o potencial

do modelo em ilha e que, para atingi-lo, é necessário atentar aos parâmetros e operadores usados, pois uma configuração inadequada pode fazer com que o DGA obtenha os mesmos ou piores resultados que o modelo tradicional.

Para trabalhos futuros, uma análise dos outros modelos paralelos, em especial o hierárquico, a utilização de uma representação diferente para os escalonamentos e a hibridização do GA com outras métodos podem prover mecanismos mais refinados e adequados para a aplicação dos GAs ao problema de *Job Shop Scheduling*.

REFERÊNCIAS

[1] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics*, 3:111–137, December 1997.

[2] D. Applegate and W. Cook. A computational study of the job-shop scheduling instance. *ORSA Journal on Computing*, 3:149–156, 1963.

[3] P. J. Fleming, Takeshi Yamada, and Ryohei Nakano. Job-shop scheduling, 1997.

[4] Sigrid Knust and Peter Brucker. *Complex Scheduling*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, February 2006.

[5] José Fernando Gonçalves, Jorge José de Magalhães, Rua Dr, Roberto Frias, Jorge José, Magalhães Mendes, and Maurício G. C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:2005, 2002.

[6] Dudy Lim, Yew-Soon Ong, Yaochu Jin, Bernhard Sendhoff, and Bu-Sung Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Gener. Comput. Syst.*, 23:658–670, May 2007.

[7] Enrique Alba and José M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Gener. Comput. Syst.*, 17:451–465, January 2001.

[8] Laurens Jan Pit. Parallel genetic algorithms. Master's thesis, Leiden University, August 1995.

[9] Yu-Kwong Kwok and Ishfaq Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47:58–77, November 1997.

[10] Heesung Lee, Sungjun Hong, and Euntai Kim. Optimal classifier design method using hierarchical fair competition model based parallel genetic algorithm. In *ICCAS-SICE, 2009*, pages 2907 –2910, aug. 2009.

[11] Enrique Alba and José M. Troya. A survey of parallel distributed genetic algorithms. *Complex.*, 4:31–52, March 1999.

[12] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[13] Byung Joo Park, Hyung Rim Choi, and Hyun Soo Kim. A hybrid genetic algorithm for the job shop scheduling problems. *Comput. Ind. Eng.*, 45:597–613, December 2003.

[14] Cesar Manuel Vargas Benitez and Heitor Silverio Lopes. Hierarchical parallel genetic algorithm applied to the three-dimensional hp side-chain protein folding problem. In *SMC'10*, pages 2669–2676, 2010.

[15] Bryant A. Julstrom. Comparing darwinian, baldwinian, and lamarckian search in a genetic algorithm for the 4-cycle problem. In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 134–138, 1999.

[16] Ron Shonkwiler. Parallel genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 199–205, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[17] Mariusz Nowostawski and Riccardo Poli. Parallel genetic algorithm taxonomy. In Lakhmi C. Jain, editor, *KES*, pages 88–92. IEEE, 1999.

[18] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles Reseaux et Systems Repartis*, 10(2):141–171, 1998.

[19] Marin Golub and Leo Budin. A new model of global parallel genetic algorithm. In *University of Paisley*, pages 363–368, 2000.

[20] ZENG Wen-hua, Yiannis Papadopoulos, and David Parker. Reliability optimization of series-parallel systems using asynchronous heterogeneous hierarchical parallel genetic algorithm. 2007.

[21] Martina Gorges-Schleuter. Asparagos a parallel genetic algorithm and population genetics. In J. Becker, I. Eisele, and F. Mündemann, editors, *Parallelism, Learning, Evolution*, volume 565 of *Lecture Notes in Computer Science*, pages 407–418. Springer Berlin / Heidelberg, 1991. 10.1007/3-540-55027-5_24.

[22] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[23] Vadim G. Timkovsky. Is a unit-time job shop not easier than identical parallel machines? *Discrete Applied Mathematics*, 85(2):149 – 162, 1998.

[24] Genetic algorithm utility library (gaul). `http://gaul.sourceforge.net/`. Acessado em: 26/11/2011.

[25] Shyh-Chang Lin, Erik D. Goodman, and William F. Punch, III. Investigating parallel genetic algorithms on job shop scheduling problems. In *Proceedings of the 6th International Conference on Evolutionary Programming VI*, EP '97, pages 383–393, London, UK, 1997. Springer-Verlag.

[26] M. Moonen and G.K. Janssens. *A Giffler-Thompson focused genetic algorithm for the static job-shop scheduling problem*. ITEO research paper series. Limburg University, Institute for Applied Economic Research, 2004.

[27] Takeshi Yamada and Ryohei Nakano. Genetic algorithms for job-shop scheduling problems. In *In Proceedings of Modern Heuristic for Decision Support*, pages 474–479. Morgan Kaufmann, 1997.

[28] Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. On permutation representations for scheduling problems. In *In 4th PPSN*, pages 310–318. Springer-Verlag, 1996.

[29] Tamer F. Abdelmaguid. Representations in genetic algorithm for the job shop scheduling problem: A computational study. *JSEA*, 3(12):1155–1162, 2010.

[30] S. C. Lin, E. D. Goodman, and W. F. Punch. A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems. In T. Bäck, editor, *Seventh International Conference on Genetic Algorithms*, pages 481–488. Morgan Kaufmann, 1997.

[31] Fatima Ghedjati. Genetic algorithms for the job-shop scheduling problem with unrelated parallel constraints: heuristic mixing method machines and precedence. *Comput. Ind. Eng.*, 37:39–42, October 1999.

[32] Manuel Vazquez and L. Darrelle Whitley. A comparison of genetic algorithms for the dynamic job shop scheduling problem. In *Genetic and Evolutionary Computation Conference, GECCO 2000*, 2000.

[33] Manuel Vázquez and Darrell Whitley. A comparison of genetic algorithms for the static job shop scheduling problem. In Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 303–312. Springer Berlin / Heidelberg, 2000. 10.1007/3-540-45356-3_30.

[34] S. M. Kamrul Hasan, Ruhul Sarker, and David Cornforth. Hybrid genetic algorithm for solving job-shop scheduling problem. *6th IEEEACIS International Conference on Computer and Information Science ICIS 2007*, (Icis):519–524, 2007.

[35] Hsiao lan Fang, Hsiao lan Fang, Peter Ross, Peter Ross, Dave Corne, and Dave Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 375–382. Morgan Kaufmann, 1993.

[36] Shigenobu Kobayashi, Isao Ono, and Masayuki Yamamura. An efficient genetic algorithm for job shop scheduling problems. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 506–511, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[37] Di-Wei Huang and Jimmy Lin. Scaling populations of a genetic algorithm for job shop scheduling problems using mapreduce. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 780–785, Washington, DC, USA, 2010. IEEE Computer Society.

[38] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *R. Manner, B. Manderick (Eds.), Parallel Problem Solving from Nature 2*, pages 281–290, North-Holland, Amsterdam, 1992.

[39] J. E. Beasley. Job shop scheduling. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html`. Acessado em: 29/11/2011.

[40] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *J. F. Muth, G. L. Thompson (Eds.), Industrial Scheduling*, pages 225–251, 1963.

[41] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, March 1988.

[42] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration*, 1984.