

**Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de
Computação**

**Disciplina: Computação Evolutiva
Professor: Fernando J. Von Zuben.**

MANUAL E RELATÓRIO DE EXPERIMENTOS

Tema: Scheduling – Problema Job-Shop

Realizado por:

Daiane Cristina Angolini	RA 002887
Francisco Osvaldo M. M. Filho	RA 981216
Pedro de Vasconcellos Castro	RA 009611
Ricardo Capitânio M. da Silva	RA 992386
Sara Luísa de Andrade Fonseca	RA 002487
Thiago Jung Bauermann	RA 002557

Campinas, Maio – 2004.

Índice

<i>Resumo</i>	3
1. INTRODUÇÃO	3
2. OBJETIVOS.....	3
3. REPRESENTAÇÃO DA SOLUÇÃO CANDIDATA.....	4
4. ARQUITETURA DO SOFTWARE	5
4.1 Operadores.....	8
Seleção por Método da Roleta.....	8
Crossover de 1 ponto	8
Crossover Uniforme	8
Mutaç�o	8
Reproduç�o Assexuada.....	9
5. MANUAL DE USO DO TOOLBOX	9
5.1 Arquivos Disponibilizados	9
5.2 Como rodar o toolbox.....	9
5.3 Dados de Entrada : Par�metros e Dados do Problema	10
5.4 Dados de Sa�da	11
6. RELAT�RIOS DE EXPERIMENTOS.....	12
6.1 Benchmarks	12
6.2 An�lise de Par�metros	12
6.3 Converg�ncia do AG	15
7. CONCLUS�ES	16
8. CONTATO COM OS AUTORES	17

Algoritmos Genéticos e o Problema Job-Shop

Resumo

Soluções de problemas envolvendo schedules – mais especificamente o Job-Shop - têm sido propostos nos últimos 40 anos. Vários modelos foram propostos para resolver o problema. Algoritmos genéticos é uma abordagem que elucida a complexidade combinatorial do problema.

1. INTRODUÇÃO

Pesquisa em teoria de scheduling evoluiu durante os últimos 40 anos, rendendo significantes descobertas, técnicas e métodos envolvendo desde rudes técnicas de despacho até algoritmos de branch and bound altamente sofisticados. Métodos têm sido formulados em um conjunto diversificado de pesquisadores, envolvendo diversas áreas de conhecimento, desde administração até pesquisa operacional e técnicas de produção. Atualmente, técnicas envolvendo redes neurais, computação evolutiva têm sido usadas para atacar o problema, o que nos dá uma dimensão da natureza multidisciplinar deste campo.

Um dos modelos mais populares na teoria de scheduling é o Job-Shop, pois ele nos dá uma boa representação do domínio geral além de ser difícil de ser resolvido. Ele é provavelmente o modelo mais estudado e bem desenvolvido em teoria de scheduling determinístico, servindo, portanto, como um comparador para teste de eficiência de diversas técnicas de solução. Além disso, esse modelo é um dos que mais se assemelha a problemas práticos de empresas de manufatura, despacho, distribuição de recursos para tarefas específicas, etc... servindo também de base para proposição de modelos mais genéricos, como distribuição de recursos a tarefas de acordo com prioridades (estáticas ou dinâmicas), utilização de recursos simultaneamente por tarefas distintas, etc...

2. OBJETIVOS

O presente documento contempla os seguintes tópicos:

- **Resolução do modelo Job-Shop:**
O escopo deste trabalho é o modelo Job-Shop. Assim, o modelo proposto, técnicas de solução, etc... foram propostos para resolver apenas esse problema clássico.
- **Uso de Algoritmos Genéticos:**
Neste toolbox usa-se apenas algoritmos genéticos para resolver o problema de JobShop. Nenhuma outra técnica foi usada (busca local, busca tabu, etc).
- **Arquitetura de software do Job-Shop Scheduling ToolBox:**
Neste documento apresenta-se, de maneira geral, a arquitetura do toolbox, com fluxogramas e explicação da representação e dos operadores usados.
- **Manual do usuário do ToolBox;**

Neste documento apresenta-se um pequeno e resumido manual do toolbox, com explicação de cada parâmetro, bem como uma explicação de como inserir os dados do problema.

- **Relatório de experimentos;**
Testes foram realizados para validar a solução proposta e verificar o quão próximo a solução obtida está da solução ótima, verificar rapidez de convergência além de outras coisas.

3. REPRESENTAÇÃO DA SOLUÇÃO CANDIDATA

A representação da solução candidata (cromossomo) utilizada neste ToolBox separa o fenótipo do genótipo. Isso significa que para obter o escalonamento correspondente a um dado indivíduo, é preciso aplicar um algoritmo (*schedule builder*) sobre o genótipo.

O genótipo é uma cadeia de $j \times m$ números inteiros, onde cada gene assume valores entre 0 a $j - 1$, onde j é o número de jobs e m é o número de máquinas. O *schedule builder* lê seqüencialmente os genes do cromossomo e para cada gene i , insere a próxima tarefa do i -ésimo job incompleto não completada no fenótipo. A lista de jobs incompletos mantida pelo algoritmo é circular, ou seja, se não houver i -ésimo job incompleto, é escolhido o job da lista circular na posição $(i \text{ modulo } n)$, onde n é o número de jobs incompletos.

Exemplo:

Considere o seguinte problema:

	(m, t)	(m, t)	(m, t)
Job 0	2, 1	0, 3	1, 2
Job 1	1, 2	3, 1	0, 2
Job 2	0, 2	1, 2	3, 2

Genótipo: 021111122

0 → primeiro job incompleto: job 0, máquina 2

2 → terceiro job incompleto: job 2, máquina 0

1 → segundo job incompleto: job 1, máquina 1

1 → segundo job incompleto: job 1, máquina 3

1 → segundo job incompleto: job 1, máquina 0 (fim do job 1)

1 → segundo job incompleto: job 2, máquina 1 (agora o job 2 é o 2º job incompleto)

1 → segundo job incompleto: job 2, máquina 1 (fim do job 2)

2 → terceiro job incompleto: job 0, máquina 0 (único job incompleto)

2 → terceiro job incompleto: job 0, máquina 1 (fim do job 0)

Por não haver uma relação direta entre o valor de um gene no cromossomo e um job no escalonamento, a representação utilizada tem alto grau de redundância, ou seja, vários genótipos podem dar origem ao mesmo fenótipo ou a fenótipos muito parecidos. Além disso, variações nas seções finais do cromossomo causam impacto menor no fenótipo (e conseqüentemente no fitness) do indivíduo que variações no início do cromossomo. Isso ocorre porque o significado dos valores dos genes no final do cromossomo depende muito dos valores dos genes que estão no início do mesmo.

4. ARQUITETURA DO SOFTWARE

No programa foi utilizada uma única instância do gerador pseudo-aleatório oferecido pela biblioteca da linguagem Java (classe `java.util.Random`). Esta instância é utilizada em todos os momentos do algoritmo em que é necessário gerar um número randômico.

O fluxograma 1 mostra o algoritmo genético adotado no toolbox. Uma descrição mais detalhada dos operadores pode ser vista ainda nesta seção.

A população inicial, de tamanho M , é criada aleatoriamente. Em seguida, seu fitness é avaliado e os indivíduos são ordenados segundo o fitness. Inicia-se então, o processo de construção da nova geração.

O melhor indivíduo da geração atual é passado diretamente para a nova geração. O algoritmo entra então em um laço para completar a nova geração, também de tamanho M . Escolhe-se dois indivíduos através do método da roleta. Estes indivíduos têm uma probabilidade p_c de sofrerem crossover. Se sofrerem crossover, têm ainda chance de sofrer crossover uniforme ou de 1 ponto (50% de chance para cada um). Se não sofrem crossover, poderão sofrer reprodução assexuada, com probabilidade p_a . Pode ser também que a reprodução não seja aplicada. Em todos os casos os indivíduos resultantes estão sujeitos a sofrerem mutação com probabilidade p_m . Os filhos gerados competem com os pais e os dois melhores indivíduos são copiados para a nova geração. Repete-se este processo até que a nova geração esteja completa.

No laço externo é necessário checar se o critério de parada foi atingido. Foi utilizado o seguinte critério de parada:

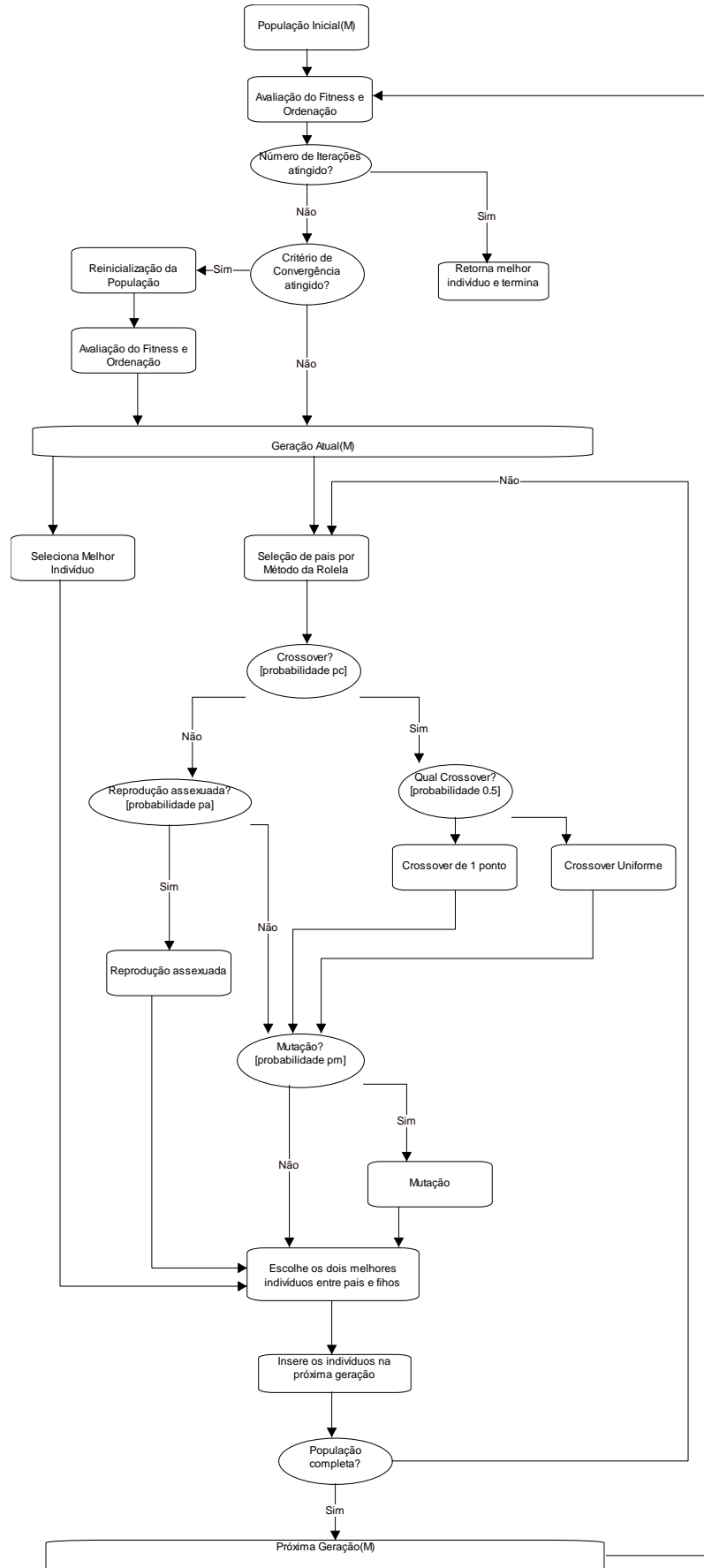
- Número máximo de iterações: é o número de gerações que o AG deve avaliar.

Também é necessário checar se esta população convergiu para valores de fitness muito parecidos, indicando que o AG atingiu um mínimo local e está com dificuldades para deixá-lo. Neste caso, 50% da população é reinicializada, ou seja, é substituída por novos indivíduos gerados aleatoriamente. O fitness dos novos indivíduos é calculado e a população é ordenada. Para indicar convergência, foi utilizado o seguinte critério:

- Número de iterações sem melhora do fitness do melhor indivíduo: este número é um parâmetro do algoritmo fornecido pelo usuário. Quando o algoritmo

permanecer com o mesmo valor de fitness para o melhor indivíduo durante este número de gerações, a população é obrigatoriamente reinicializada.

Se o AG não pára, tendo ou não reinicializado a população, reinicia-se o processo de construção da nova geração.



Fluxograma 1 – Algoritmo Genético Proposto

4.1 Operadores

Seleção por Método da Roleta

Na seleção de indivíduos por método da roleta, um vetor de intervalos é construído a partir do fitness de cada indivíduo. A primeira posição tem o valor do fitness do primeiro indivíduo. As posições seguintes têm o valor da posição anterior mais o fitness do indivíduo correspondente. O intervalo do *i*-ésimo indivíduo é o seu valor correspondente no vetor menos o anterior. Desta forma, indivíduos com fitness maiores, terão um intervalo maior. Um número é gerado aleatoriamente entre 0 e o último valor do vetor. O indivíduo cujo intervalo contém este número é o escolhido.

Crossover de 1 ponto

O crossover de 1 ponto escolhe aleatoriamente uma posição entre 0 e $0,7 \cdot \text{tamanho do indivíduo}$. O tamanho do indivíduo será o produto entre número de jobs e número de máquinas. Adotou-se este critério devido à ambigüidade da representação adotada: a partir de uma certa posição, os valores dos genes passam a depender da configuração dos genes anteriores. Desta forma, não haveria modificação significativa se apenas a parte final do cromossomo fosse modificada. O esquemático abaixo mostra como isso é feito, para um problema de três máquinas e 3 jobs:

$$\begin{array}{ccc}
 [1\ 2\ 3\ 1\ 1 \parallel 2\ 3\ 3\ 2] & + & [1\ 1\ 1\ 2\ 2 \parallel 2\ 3\ 3\ 3] \\
 \downarrow & & \downarrow \\
 [1\ 2\ 3\ 1\ 1\ 2\ 3\ 3\ 3] & & [1\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 2]
 \end{array}$$

Crossover Uniforme

O crossover uniforme cria dois novos indivíduos a partir de dois pais. Cada filho herda o gene de um ou outro pai segundo uma probabilidade de 50%. A troca é feita apenas até 70% do cromossomo, já que a partir daí, o código do cromossomo torna-se redundante e a troca não significativa. O esquemático abaixo mostra como isso é feito, para um problema de três máquinas e 3 jobs:

$$\begin{array}{ccc}
 [1\ 2\ 3\ 1\ 1\ 2 \mid 3\ 3\ 2] & + & [1\ 1\ 1\ 2\ 2\ 2 \mid 3\ 3\ 3] \\
 \downarrow & & \downarrow \\
 [1\ 1\ 3\ 1\ 1\ 2\ 3\ 3\ 2] & & [1\ 2\ 1\ 2\ 2\ 2\ 3\ 3\ 3]
 \end{array}$$

Mutação

A mutação é realizada trocando-se um dos genes do cromossomo por outro escolhido aleatoriamente dentre os possíveis valores. O gene que será trocado é escolhido aleatoriamente entre 0 e 70% do cromossomo, também pelos mesmos motivos citados anteriormente.

[1 2 3 1 1 2 3 3 2]



[1 1 3 1 1 2 3 3 3]

Reprodução Assexuada

Este operador leva o nome de assexuado porque é realizado sobre um único indivíduo. O novo indivíduo é formado escolhendo-se aleatoriamente uma posição do cromossomo e trocando as duas partes. O esquemático abaixo mostra como funciona o operador:

[1 2 3 1 1 || 2 3 3 2]



[2 3 3 2 1 2 3 1 1]

5. MANUAL DE USO DO TOOLBOX

5.1 Arquivos Disponibilizados

Os seguintes arquivos compõem o toolbox desenvolvido:

- s-tool.jar : arquivo executável contendo o algoritmo genético.
- Pacotes Java: br.org.codigolivre.gui (classes de interface com o usuário) e br.org.codigolivre.ga (classes da solução utilizando ga)
- benchmarks.zip: instâncias de teste detalhadas neste manual.

Estes arquivos podem ser encontrados no endereço <http://s-tool.codigolivre.org.br>.

5.2 Como rodar o toolbox

Para fins de execução do tool box, o usuário deve ter instalado na estação de trabalho a máquina virtual java. O arquivo "executável" possui extensão ".jar". No diretório onde o arquivo ".jar" estiver, o usuário deve digitar a seguinte linha de comando:

```
java -jar s-tool.jar
```

onde "s-tool" se refere ao nome do arquivo. Uma vez feito isso, a interface mostrada na figura 1 a seguir aparece.

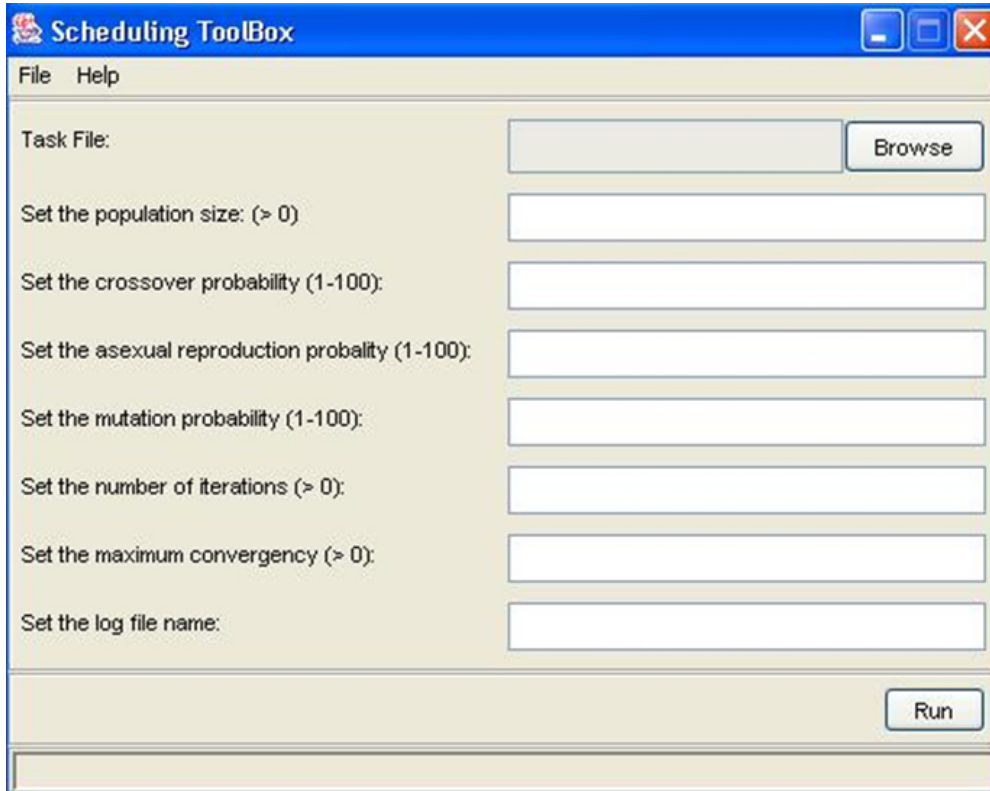


Figura 1: Interface inicial

5.3 Dados de Entrada : Parâmetros e Dados do Problema

Nesta tela, o usuário deve inserir todos os parâmetros (todos são obrigatórios). Alguns dos parâmetros são de fácil compreensão, outros nem tanto:

- **Task File:** Através do botão *browse* deve-se escolher o arquivo de entrada contendo os dados do problema. O arquivo de entrada deve possuir a formatação a seguir:

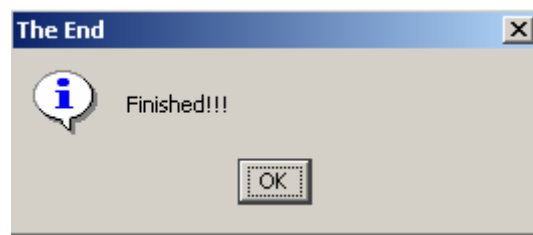
1° linha: $j \ m$ // j é o número de jobs e m é o número de máquinas
 2° linha: $(m_1, t_1) (m_2, t_2) (m_3, t_3) \dots (m, t_m)$ // lista de tarefas do primeiro job
 3° linha: $(m_1, t_1) (m_2, t_2) (m_3, t_3) \dots (m, t_m)$ // lista de tarefas do segundo job
 4° linha: $(m_1, t_1) (m_2, t_2) (m_3, t_3) \dots (m, t_m)$ // lista de tarefas do terceiro job

- **Population Size:** Define o tamanho da população de indivíduos em cada geração. Cada indivíduo é um cromossomo codificado como mostrado na seção 3 deste documento.
- **Crossover Probability (pc):** Define a probabilidade de ocorrer crossover entre dois indivíduos.
- **Asexual Reproduction Probability (pa):** Define a probabilidade de ocorrer reprodução assexuada em cada indivíduo.

- **Mutation Probability (pm):** Define a probabilidade de ocorrer mutação em cada indivíduo.
- **Number of Iterations:** Define o número de gerações (ou o número de evoluções) a partir de uma determinada população inicial.
- **Maximum Convergency:** Define o número máximo de gerações sem melhora da solução ótima antes de inserir uma perturbação aleatória na população. Maiores detalhes na seção 4.
- **Log File Name:** Define o nome do arquivo de log para onde os dados relevantes de cada geração serão exportados.

Uma vez carregado o arquivo de entrada, o usuário está apto a iniciar a execução do algoritmo genético, para isso clicando no botão *run*.

Assim que a execução do algoritmo genético tiver terminado, a seguinte tela aparece:



5.4 Dados de Saída

Uma vez terminada a execução do algoritmo genético, o usuário pode ir ao diretório onde está o arquivo “executável” com extensão “.jar” e procurar pelo arquivo de log gerado. O formato do arquivo de log é mostrado a seguir:

```

1º linha: I average fitness and best fitness of initial population
2º linha: 0 average fitness and best fitness of next generation
3º linha: 1 average fitness and best fitness of next generation
4º linha: 2 average fitness and best fitness of next generation
...
n-ésima linha: n average fitness and best fitness of last generation
n+1-ésima linha: fenotipe and fitness of the best individual

```

Algumas vezes pode aparecer uma linha com a frase “Convergency Reached”, o que significa que a população ficou em uma solução “ótima” fixa durante n gerações, onde n é o parâmetro “Maximum convergency” descrito anteriormente. Quando essa situação acontece, metade da população é descartada (somente os piores indivíduos) e substituída por novos indivíduos criados aleatoriamente.

6. RELATÓRIOS DE EXPERIMENTOS

6.1 Benchmarks

Durante os testes foram utilizadas algumas das instâncias de benchmarks propostos originalmente por Taillard (1993) para o problema de jobshop. Os benchmarks contêm 80 instâncias do problema, com número de jobs entre 15 e 100 e número de máquinas entre 15 e 20. Todos os tempos de operação são gerados randomicamente em um intervalo único, variando entre 1 e 99 unidades de tempo. As instâncias estão disponíveis na ORLIB (Operational Research Library), no endereço <http://mscmga.ms.ic.ac.uk/info.html>.

Os benchmarks também foram disponibilizados pelo grupo no arquivo benchmarks.zip. Este arquivo contém as 80 instâncias, formatadas segundo a entrada do toolbox. Os arquivos são nomeados `taxx`, onde `xx` é o número da instância. Juntamente com estes arquivos está o arquivo `best_lb_up.txt` que contém os limites inferiores teóricos (lower bound) e a melhor solução encontrada até então.

6.2 Análise de Parâmetros

Nesta seção, a sensibilidade a variação dos parâmetros do AG foi analisada. Procedeu-se fixando todos os parâmetros em valores considerados razoáveis em testes preliminares, e variando o parâmetro em análise em um certo intervalo.

As seguintes observações são feitas para os gráficos apresentados:

- Foi utilizada a instância `ta02`, com número de jobs 15 e número de máquinas 15. O melhor makespan encontrado até então na literatura é de 1244 unidades de tempo. É importante lembrar que a melhor solução para este problema foi encontrada por uma aplicação de Busca Tabu.
- Os resultados indicados correspondem a média de três execuções do algoritmo.
- Para todos os gráficos foram usados os seguintes parâmetros mostrados a seguir. Entretanto, o parâmetro analisado em cada gráfico assumiu valores dentro do intervalo indicado nos gráficos.
 - Tamanho da população: 700
 - Probabilidade de reprodução assexuada: 12%
 - Probabilidade de crossover: 65%
 - Probabilidade de mutação: 4%
 - Número de gerações: 1400
 - Convergência máxima: 14

Os resultados obtidos pelo uso do toolbox são apresentados a seguir:

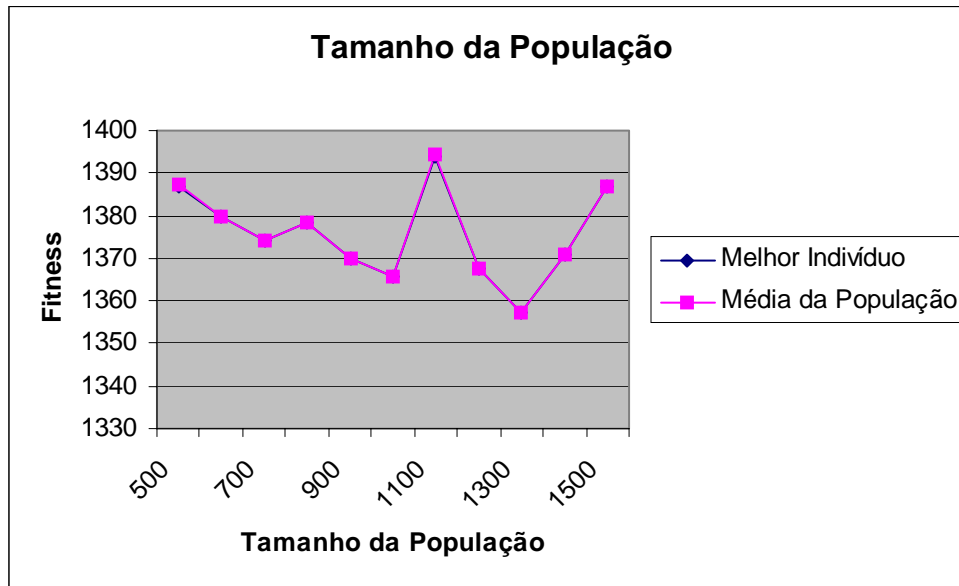


Gráfico 1: Parâmetro em análise: Tamanho da população

A princípio, pensou-se que quanto maior a população maior seria a diversidade da população, o que implicaria em um espaço de busca maior e, com isso, poder-se-ia obter uma solução mais próxima da solução ótima. No entanto, mantendo o número de gerações constante, observa-se, pela análise do gráfico 1, que a variação do tamanho da população não melhorou o resultado obtido.

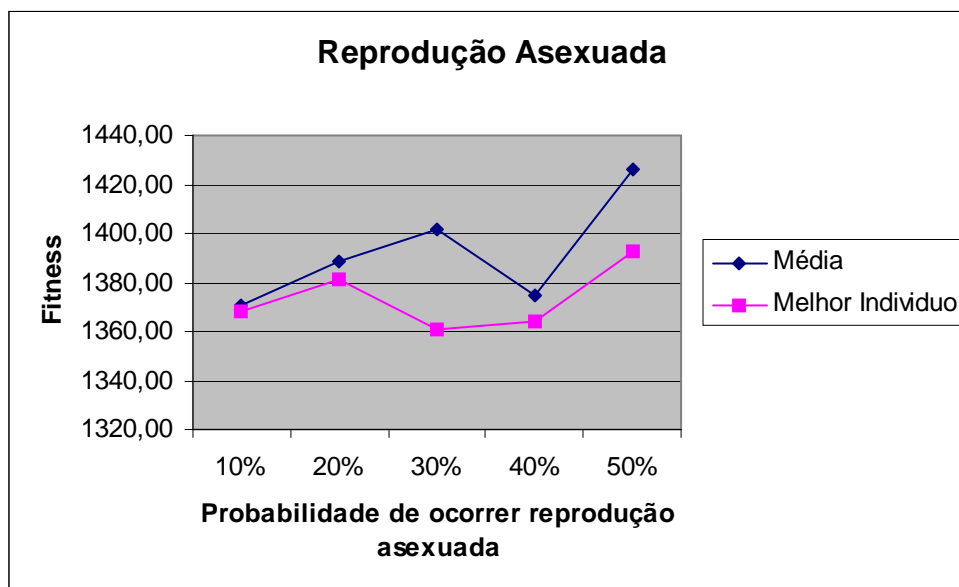


Gráfico 2: Parâmetro em análise: Probabilidade de Reprodução Asexual

O operador de reprodução assexual foi proposto e implementado com o intuito de diversificar os indivíduos de uma população caso o crossover não ocorresse. Uma suposição a que pode-se chegar analisando o gráfico 2 é a de que para valores de probabilidade de reprodução assexual entre 30% e 40% obteve-se os melhores resultados. Infelizmente não podemos generalizar essa conclusão, pois testou-se o toolbox apenas para o benchmark TA02.

Seria necessário avaliar essa suposição para outros problemas e observar os melhores resultados obtidos em cima desses problemas variando essa probabilidade para poder tirar conclusões mais concretas.

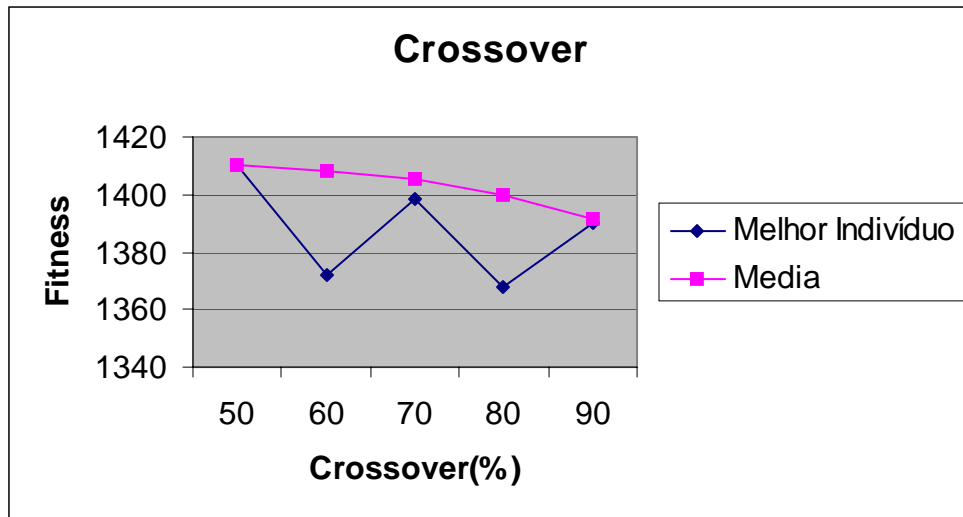


Gráfico 3: Parâmetro Analisado: Probabilidade de Ocorrer Crossover

Analisando o gráfico 3 podemos perceber que a média da população tende a diminuir com o aumento da probabilidade de crossover. O fitness do melhor indivíduo tende a oscilar em torno de valores menores com o aumento da probabilidade de crossover. Desta forma, pode-se concluir que taxas mais altas para crossover devem ser consideradas em detrimento de taxas pequenas.

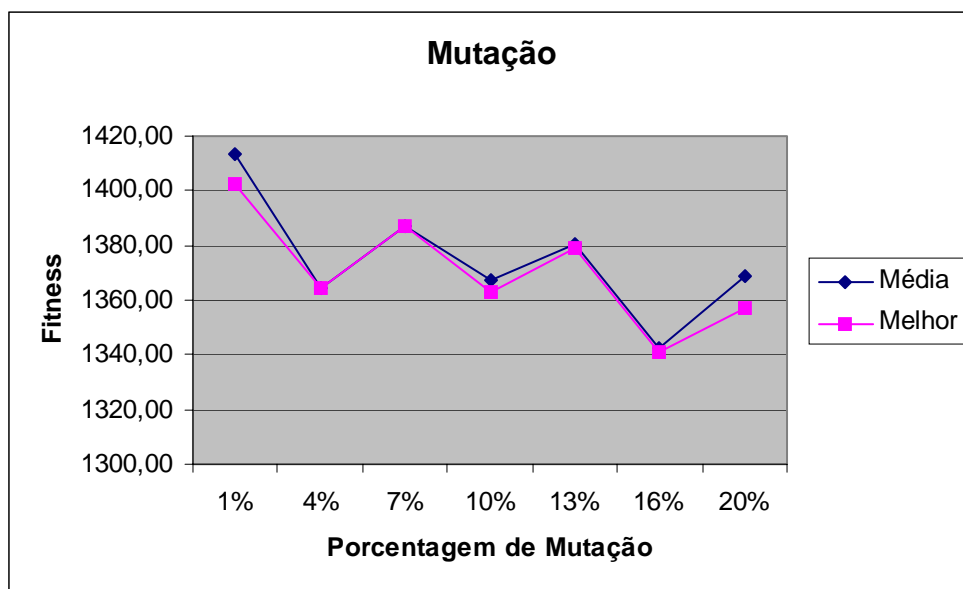


Gráfico 4: Parâmetro Analisado: Probabilidade de Ocorrer Mutação

O gráfico acima foi gerado com a finalidade de estudar as alterações provocadas pelo parâmetro de mutação sobre o algoritmo genético implementado. Observa-se uma tendência de melhora no indivíduo de melhor Fitness. Entretanto, há uma certa oscilação em torno de uma reta média, que tende a aumentar com aumento exacerbado desse parâmetro devido a características aleatórias.

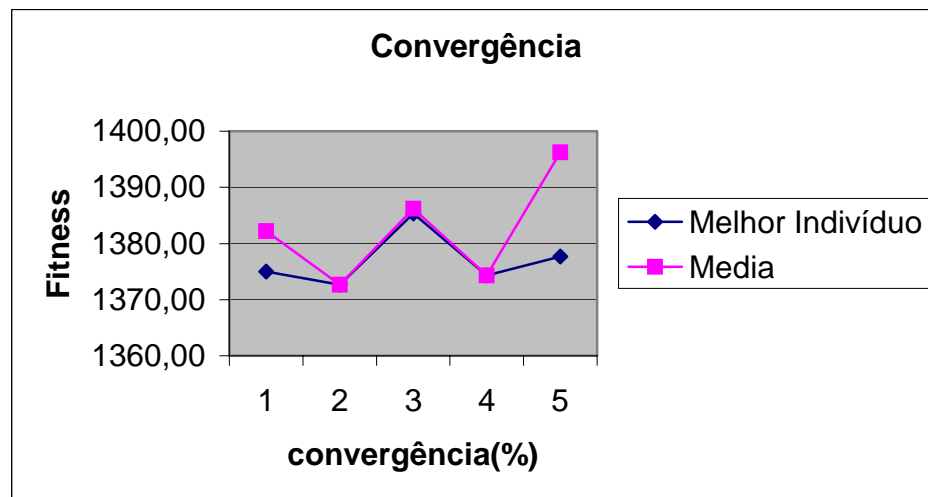


Gráfico 5: Parâmetro Analisado: Critério de Convergência

A partir do gráfico 5 pode-se observar que este parâmetro, se utilizado entre 1 e 5 % não afetará consideravelmente o desempenho do algoritmo. Deve-se lembrar que em testes preliminares utilizou-se altas taxas para este parâmetro (>10%) e os resultados foram não satisfatórios.

6.3 Convergência do AG

Para o estudo da velocidade de convergência do algoritmo realizou-se três experimentos mantendo-se os mesmos parâmetros (População de 700 indivíduos, probabilidade de crossover de 65%, probabilidade de reprodução assexuada de 12%, probabilidade de mutação de 4% e número de iterações de 1400, e inserção de novos indivíduos na população que convergiu a 14 iterações). O gráfico 6 mostra o fitness do melhor indivíduo ao longo das iterações para os três experimentos. O gráfico 7 é idêntico ao gráfico 6, mas com um menor número de iterações, para observar mais detalhadamente a convergência no início das iterações.

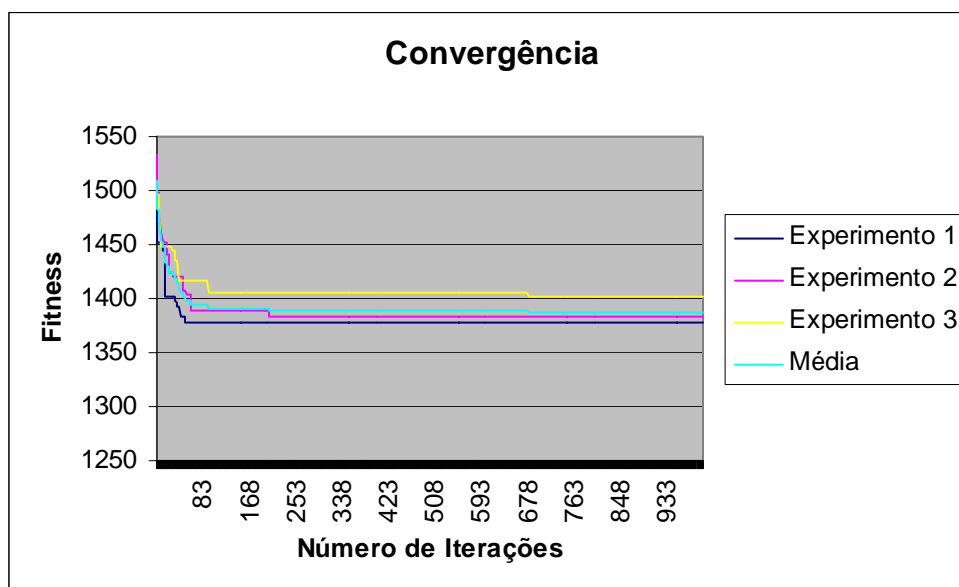


Gráfico 6: Convergência em 1000 iterações

Pelo gráfico 6, pode-se observar que a convergência em todos os experimentos se dá de forma bastante rápida: em torno de 100 iterações todos os experimentos convergiram. Após isso, com a perda da diversidade os experimentos até podem encontrar uma solução ligeiramente melhor (em termos de unidade de tempo do makespan), mas isso é praticamente o uso de força bruta em uma busca praticamente aleatória. O gráfico 7 a seguir (ampliado) mostra a rápida convergência do algoritmo:

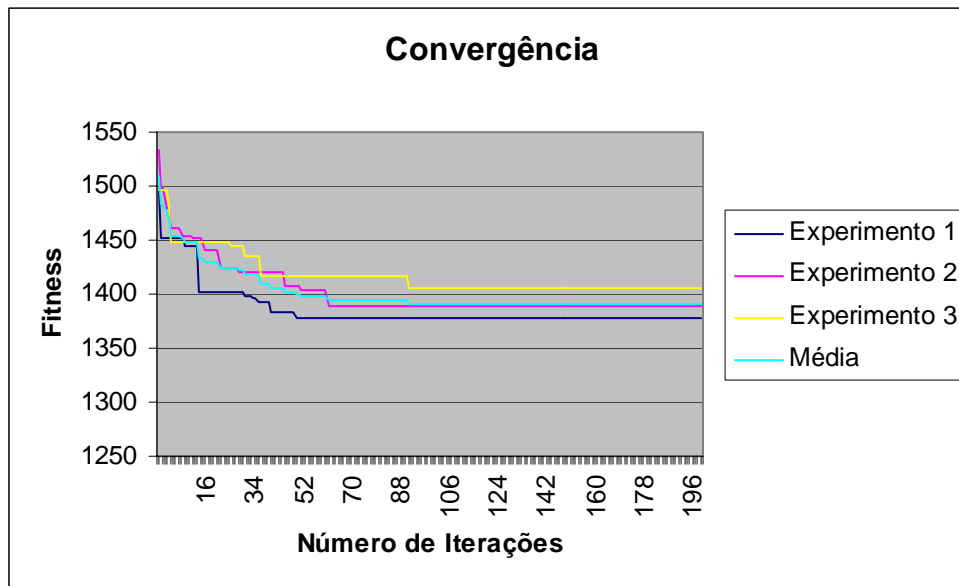


Gráfico 7: Convergência em 200 iterações

7. CONCLUSÕES

Os testes realizados para a análise de parâmetros do algoritmo podem ser aprofundados e um estudo pode ser feito para outras instâncias de problemas. Entretanto, os experimentos executados cumprem o papel de indicar ao usuário como os parâmetros devem ser analisados e sugerem a ordem dos valores a serem utilizados para cada parâmetro.

A convergência rápida do algoritmo pode ser parcialmente atribuída à redundância inerente à representação adotada, que diminui o poder de busca do algoritmo.

A representação utilizada dá a liberdade para o uso dos operadores clássicos dos algoritmos genéticos. Entretanto, devido à geração de soluções muito parecidas em termos de makespan a partir de genótipos totalmente diferentes leva a uma exploração não tão eficiente do espaço de busca. Um estudo maior dessa característica e a geração de operadores especialmente projetados para essa representação podem levar a resultados mais satisfatórios.

Observando-se os melhores resultados apresentados nos benchmarks, percebe-se a presença de muitas técnicas de busca local, como a Busca Tabu, entre outras, e combinações de heurísticas.

Este fato, combinado à rápida convergência do AG detectada, leva à conclusão de que a hibridização do AG proposto poderia ajudar na procura de mínimos globais (deixando mínimo locais) e, portanto, levar a resultados mais promissores.

8. CONTATO COM OS AUTORES

Para quaisquer dúvidas que possam restar sobre algum ponto da implementação ou sobre o uso do toolbox, o grupo pode ser consultado através do e-mail:

jobshop_GA@yahoogroups.com.br

Os componentes do grupo podem ser consultados individualmente através dos e-mails pessoais:

Daiane Cristina Angolini

dai_ang@yahoo.com.br

Francisco Osvaldo M. M. Filho

chico_osvaldo@yahoo.com.br

Pedro de Vasconcellos Castro

ra009611@ic.unicamp.br

Ricardo Capitano M. da Silva

ricardo.capitania@ic.unicamp.br

Sara Luísa de Andrade Fonseca

saraluisa_fonseca@yahoo.com.br

Thiago Jung Bauermann

ra002557@ic.unicamp.br