Um Algoritmo Genético aplicado ao Problema de *Job Shop* Clássico: Estudo Comparativo entre os Operadores de *Crossover*

Antonio Costa de Oliveira, Filipe de Oliveira Saraiva

Departamento de Informática e Estatística - Universidade Federal do Piauí
Campus Ministro Petrônio Portella, Centro de Ciências da Natureza
SG-09 CEP 64.049-550 Teresina - Piauí

costa@ufpi.br, filipe.saraiva@ymail.com

Abstract. *The Job Shop Problem is a combinatorial optimization problem with a wide practical application in the manufacturing optimization. Because of the great number of possible combinations between decision variables, obtaining a optimum solution for the problem by precise methods is normally impracticable given the high computational cost. However, many researchers use a heuristic abordation between them, the Genetic Algorithms, where is intended to find a good solution with a reasonable computational cost. This paper makes a comparative analysis between five genetic crossover operators applied in a Genetic Algorithm for the Job Shop Problem, analyzing their results and performance for the proposed instances.*

Resumo. O Problema de *Job Shop* é um problema de otimização combinatória com vasta aplicação prática na otimização da produção de empresas e indústrias. Por conta do grande número de combinações entre suas variáveis de decisão, a obtenção da solução ótima ao problema por métodos exatos é normalmente inviável dado o alto custo computacional inerente. Porém, vários pesquisadores utilizam abordagens heurísticas, entre elas, os Algoritmos Genéticos, para encontrar uma boa solução a custo computacional hábil. Este trabalho faz um estudo comparativo entre operadores genéticos de *crossover* aplicados a um Algoritmo Genético para o Problema de *Job Shop* Clássico, analisando seus resultados e desempenho para as instâncias propostas.

1. Introdução

Problemas de otimização combinatória são problemas onde busca-se encontrar um arranjo de variáveis de decisão, representadas por objetos discretos, em que a solução encontrada represente a solução ótima ao problema proposto (Campello e Maculan, 1994; Reeves, 1993).

A principal dificuldade em se tratar problemas de otimização combinatória reside no grande número de combinações existentes entre as variáveis de decisão que compõem o modelo do problema. Esse número de combinações resulta num espaço de busca muito grande, impossibilitando a aplicação eficaz de métodos de resolução exatos a um custo computacional razoável (Reeves, 1993).

Apesar desta característica, vários pesquisadores estudam métodos de resolução de problemas de otimização combinatória baseada em modelos heurísticos. Heurísticas são métodos que procuram boas soluções, não necessariamente a ótima, a um custo computacional razoável (Norvig e Russel, 2004). As pesquisas nessa área evoluíram os

conceitos e implementações destes métodos, de forma que eles tornaram-se mais robustos e versáteis para aplicações a problemas desse tipo.

Este artigo trata da aplicação de um algoritmo genético ao problema de *job shop* clássico, fazendo um estudo comparativo entre cinco operadores de *crossover* encontrados na literatura. A seção 2 define o problema estudado e a seção 3 a teoria sobre algoritmos genéticos . A seção 4 apresenta as características do algoritmo genético utilizado, bem como os operadores de *crossover* comparados. A seção 5 apresenta as instâncias e resultados computacionais e, por fim, a seção 6 trás as conclusões do estudo e as possibilidades de trabalhos futuros.

2. Problema de *Job Shop* Clássico

O problema de programação de tarefas em um ambiente *job shop* pode ser definido por um conjunto de *m* máquinas e um conjunto de *n* tarefas, onde cada tarefa consiste de uma sequência ordenada de *o* operações. Para o problema de *job shop* clássico, objeto deste estudo, o número de operações será igual ao número de máquinas *(o = m).*

Cada operação deve ser processada em uma única máquina por um determinado tempo sem interrupção e uma máquina pode processar somente uma operação por vez. As operações de uma mesma tarefa devem ser executadas em máquinas diferentes e cada tarefa possui uma ordem particular de processamento nas máquinas. O problema consiste em programar as operações das tarefas de forma a otimizar a medida de desempenho conhecida como *makespan* – o tempo mínimo em que todas as tarefas estarão concluídas. (Baker, 1974).

3. Algoritmo Genético

Algoritmos genéticos estão entre as heurísticas mais estudadas e desenvolvidas nas últimas décadas (Coley, 1999). Criada por John Holland no final da década de 60, baseia-se na dinâmica natural das espécies para fazer evoluir um conjunto de soluções para um determinado problema.

A representação de soluções em um algoritmo genético é feita através de uma *string* de *bits*, chamada de cromossomo, onde cada *bit* é comumente chamado de gene. Existem vários tipos de codificação, desde cromossomos de inteiros, números reais e variáveis binárias (Linden, 2006).

Dada uma população de soluções em um algoritmo genético, faz-se um processo de seleção entre suas representações para escolher quais delas passarão pelo processo de cruzamento, onde seus *bits* serão permutados entre si, a partir de uma determinada regra.

A próxima etapa será o cruzamento, ou *crossover*, entre os indivíduos selecionados na etapa anterior. A ideia é que os bons genes da população possam se combinar (Carvalho, Braga e Ludemir, 2003), formando indivíduos melhores adaptados ao ambiente – ou seja, que tenham um melhor desempenho na função objetivo.

Em seguida inicia-se o processo de mutação. O operador de mutação serve para colocar na população aqueles genes que, porventura, se perderam durante o processo de cruzamento e formação de gerações, tornando a busca mais diversificada. Quando um gene é selecionado, o valor dele é permutado para outro valor possível relacionado a codificação utilizada. Por exemplo, na codificação binária existem apenas duas possibilidades de valores para os genes, *0*'s e *1*'s. Um gene que tenha o valor *0*, se for mutado, passará a ter o valor *1*, e vice-versa.

A partir dessas fases, têm-se novas soluções que farão parte da próxima geração do problema, repetindo o ciclo até um número de gerações pré- estabelecido ou outro critério de parada, como tempo máximo de execução do método, não observação de alteração no desempenho médio da população após um dado número de gerações, e outras.

4. Características do Algoritmo Genético Implementado

No Algoritmo Genético implementado, utiliza-se a representação inteira codificada em lista de preferência (Chen, Gen e Tsujimura, 1996). O critério de seleção utiliza o método da roleta viciada (Linden, 2006).

Optou-se pelo uso do operador de inversão como forma de mutação (Croce, Tadei e Volta, 1995). No operador de inversão, selecionam-se dois genes do cromossomo e inverte-se a subcadeia de genes compreendidos entre estes dois pontos.

O critério de parada utilizado no presente trabalho foi o número máximo de gerações no valor de 1000 iterações.

As demais características do algoritmo genético implementado são: valor da taxa de mutação em 0.01; taxa de *crossover* em 0.9. A população inicial é gerada de forma aleatória, e os 5 melhores indivíduos de cada geração são mantidos na próxima, em um processo conhecido como elitismo (Coley, 1999). O tamanho da população é fixado no valor de 100 indivíduos.

Os operadores de *crossover* utilizados neste algoritmo genético foram o *PMX*, *OX*, *CX*, *LOX*, e *PPX*. Estes operadores foram retirados da literatura onde os mesmos foram aplicados com resultados satisfatórios a vários problemas de otimização combinatória, como ao Problema do Caixeiro Viajante (Michalewicz, 1996), Problema de *Job Shop* com Atraso (Mattfeld e Bierwirth, 2004), Problema de *Job Shop* Clássico (Croce, Tadei e Volta, 1995) entre outros.

4.1. PMX – *Partially Mapped Crossover*

O operador *PMX* foi proposto por Goldberg e Lingle (Michalewicz, 1996) para o Problema do Caixeiro Viajante. Dados dois cromossomos pais *p1* e *p2*, realizam-se dois pontos de corte aleatórios sobre os mesmos. As subcadeias de genes encontradas entre estes cortes serão herdadas integralmente pelos indivíduos filhos *f1* e *f2*.

Estas subcadeias também determinam um mapeamento de relacionamento entre os genes dos pais afim de tratar a inviabilidade ocasionada pelo processo. Tomemos a Figura 1 como exemplo:

p1: 1 2 | 3 4 6 | 5 → *f1:* 3 6 | 1 4 2 | 5

p2: 6 3 | 1 4 2 | 5 → *f2: 2* 1 | 3 4 6 | 5

Figura 1: Crossover PMX

4.2. OX – *Order Crossover*

O operador *OX* foi proposto por Davis (Michalewicz, 1996) também para o Problema do Caixeiro Viajante. Assim como no *PMX*, dados dois cromossomos pais *p1* e *p2*, realizar-se-ão dois pontos de corte aleatórios sobre os mesmos, onde as subcadeias

de genes encontradas entre estes cortes serão herdadas integralmente pelos indivíduos filhos *f1* e *f2*. A partir do último corte em cada cromossomo, o método faz uma busca no cromossomo do outro indivíduo pai pelos genes que não estão na subcadeia herdada, preenchendo assim o cromossomo. A Figura 2 traz um exemplo:

p1: 1 2 | 3 4 6 | 5 → *f1:* 2 1| 3 4 6 | 5
p2: 5 3 | 2 1 4 | 6 → *f2:* 3 6| 2 1 4 | 5

Figura 2: *Crossover OX*

4.3. CX – *Cycle Crossover*

O operador *CX* foi proposto por Oliver (Michalewicz, 1996) e aplicado ao Problema do Caixeiro Viajante. Ele trabalha sobre um subconjunto de genes que ocupam um mesmo conjunto de posições em ambos os pais. Estes genes são então copiados para um determinado filho, enquanto os outros genes são copiados do outro pai (Costa e Anderson, 2006). A Figura 3 apresenta um exemplo:

p1: 1 **2 3** 4 6 5 → *f1:* 5 **2 3** 1 4 6
p2: 5 **3 2** 1 4 6 → *f2:* 1 **3 2** 4 5 6

Figura 3: *Crossover CX*

4.4. LOX – *Linear Order Crossover*

O operador *LOX* é uma variação do operador *OX* e foi desenvolvido por Falkenauer e Bouffouix (Croce, Tadei e Volta, 1995). Neste operador, dois pontos de corte aleatórios são realizados nos pais. Os genes pertencentes a subcadeia de um pai são retirados do outro pai, e os espaços vazios resultantes são reunidos entre os cortes. Após esta etapa, troca-se a subcadeia anterior entre os pais, gerando os filhos. Veja o exemplo da aplicação deste operador na Figura 4 abaixo:

p1: 1 2 |3 4| 6 5 → h h 3 4 6 5 → 3 4 h h 6 5 → 3 4 2 1 6 5
p2: 5 3 |2 1| 4 6 → 5 h 2 1 h 6 → 5 2 h h 1 6 → 5 2 3 4 1 6

Figura 4: *Crossover LOX*

4.5. PPX - *Precedence Preservative Crossover*

O operador *PPX* conserva a ordem de precedência dos genes dos pais (Mattfeld e Bierwirth, 2004). Através da geração de uma máscara similar a gerada para o *crossover* uniforme (Carvalho, Braga e Ludemir, 2003), o método vai selecionando genes de um dos pais. Caso a seleção daquele gene torne o filho inviável, o operador busca o próximo gene do mesmo pai, desde que a viabilidade do filho seja mantida.

Abaixo, na Figura 5, temos o exemplo de aplicação do *PPX*. Na máscara, para geração de *f1*, os valores *0* indicam a busca de gene do *p1;* os valores *1* indicam busca de gene no cromossomo *p2*. Para geração de *f2* os valores de *0* e *1* são invertidos.

Máscara: 0 1 0 0 1 0

p1: 1 2 3 4 6 5 → *f1:* 1 6 2 3 4 5

p2: 6 3 1 4 2 5 → *f2:* 6 1 3 4 2 5

Figura 5: *Crossover PPX*

5. Resultados Computacionais

As instâncias do problema proposto foram retiradas da literatura, e compreendem alguns dos problemas encontrados na OR-Library, um repositório de várias instâncias de problemas de otimização combinatória. A Tabela 1 abaixo apresenta os tipos de problemas com seus respectivos valores no número de máquinas e tarefas:

Tabela 1 – Instâncias do Problema de *Job Shop* Clássico

Problemas		
Instâncias	**Nº de Tarefas**	**Nº de Máquinas**
ft06	6	6
abz5	10	10
abz8	20	15
yn1	20	20
swv20	50	10

Cada *crossover* implementado no algoritmo genético foi executados 5 vezes para cada instância, de forma que a média e o melhor resultado dessas execuções pudesse ser obtido. A Tabela 2 apresenta as médias (em azul na coluna M.) e melhores resultados (na coluna M.R.) para as instâncias tomadas como exemplo. Os números apresentados representam o *makespan*, ou o tempo mínimo para que todas as tarefas daquela instância estarão concluídas. A unidade de tempo, neste caso, é indefinida – assume-se, para efeito comparativo, que todas as soluções encontradas utilizam uma mesma medida de tempo.

Tabela 2: Médias e Melhores resultados para as instâncias abordadas

	ft06		**abz5**		**abz8**		**yn1**		**swv20**	
	M.	**M. R.**	**M.**	**M. R.**	**M.**	**M. R.**	**M.**	**M. R.**	**M.**	**M. R.**
PMX	55	55	1361,2	1332	792,2	792	1081,6	1069	2879,8	2843
OX	57	55	1390,2	1361	806,8	805	1102,6	1094	2928	2928
CX	58,8	55	1417	1399	808	808	1124,6	1123	2928	2928
LOX	55	55	1339,6	1301	795	787	1072,4	1064	2864,8	2838
PPX	55	55	1363,6	1337	808	808	1118,4	1097	2928	2928

As figuras abaixo mostram a relação entre as médias para os *crossovers* abordados neste estudo.

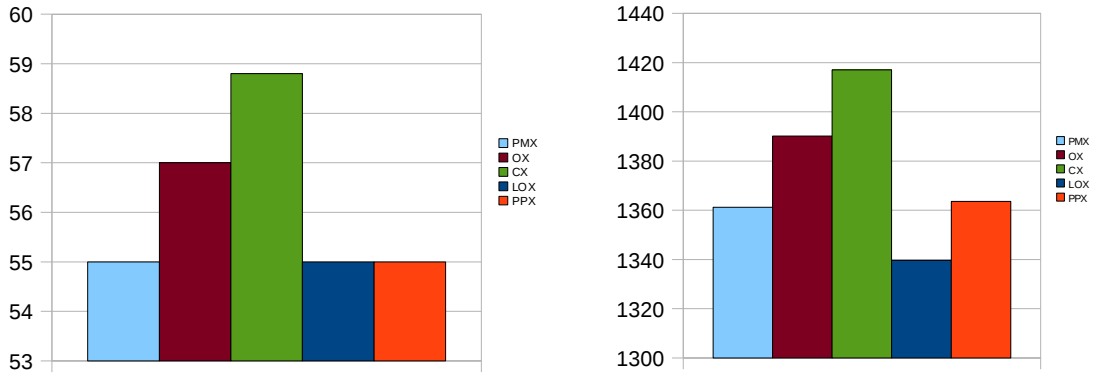

Figura 6 – Gráfico para a instância ft06 (esquerda) e abz5 (direita)

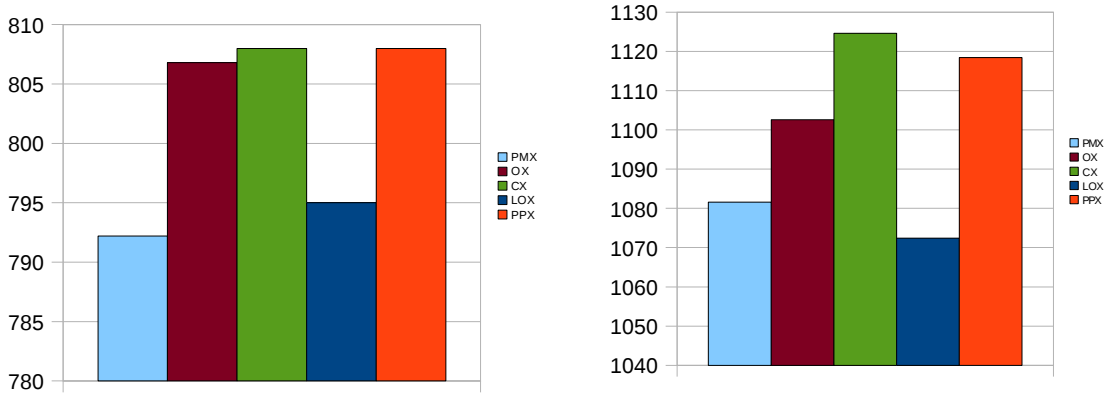

Figura 7 – Gráfico para a instância abz8 (esquerda) e yn1 (direita)

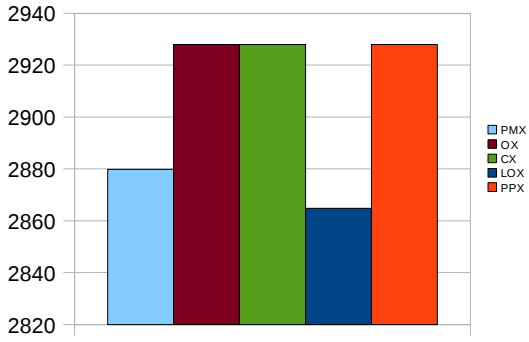

Figura 8 – Gráfico para a instância swv20

Os resultados apresentados na Tabela 2 mostram que os resultados encontrados pelo algoritmo genético para os diversos tipos de *crossover* diferem bastante em termos de qualidade quando se compara o resultado médio encontrado.

Os operadores que obtiveram melhores resultados foram o *PMX* e o *LOX*. Apenas para a instância abz8 o operador *PMX* encontrou um valor melhor otimizado que *LOX*. Nas demais instâncias, o operador *LOX* mostrou-se mais versátil na busca de melhores resultados.

Já os operadores *CX*, *OX* e *PPX* não encontraram resultados tão interessantes quanto os *LOX* e *PMX*. Percebe-se que aqueles três operadores, aplicados com bastante sucesso em determinados problemas de otimização combinatória, não são tão atrativos para a implementação de algoritmos genéticos ao problema de *job shop* clássico quanto os dois últimos.

6. Conclusões e Trabalhos Futuros

O desenvolvimento de métodos heurísticos tem grande importância para a resolução de problemas de otimização combinatória. Enquanto um algoritmo exato com custo computacional razoável não é desenvolvido, se for possível de ser desenvolvido, métodos heurísticos vão se tornando cada vez mais versáteis e robustos.

Este artigo propôs-se a realizar um estudo comparativo entre operadores de *crossover* para um algoritmo genético aplicado ao problema de *job shop* clássico. Os resultados mostraram um desempenho melhor do algoritmo com *crossover LOX* implementado, apesar do *PMX* ter tido um desempenho melhor para uma das instâncias.

Dada a dimensão e importância que os problemas de otimização combinatória tem para instituições e organizações contemporâneas, o estudo e uso de métodos heurísticos aplicados a estes problemas se mostram uma abordagem interessante e possibilitam a obtenção de boas soluções, aceitáveis para o modelo frente ao custo computacional intratável de algoritmos exatos. Assim, espera-se que o estudo aqui realizado e documentado possa servir de contribuição ao desenvolvimento de métodos cada vez mais eficientes.

Em adição, quando algum pesquisador for estudar o desempenho de algoritmos genéticos a Problemas de *Job Shop* em casos reais, já poderá utilizar os resultados aqui obtidos e projetar um algoritmo genético mais robusto e eficiente, com a utilização de operadores de *crossover* que encontram soluções melhores para a instância dada.

Como trabalho futuro, espera-se hibridizar o algoritmo genético com alguma método heurístico de busca local, como a busca tabu. A premissa é que, enquanto o algoritmo genético faz uma otimização mais global, um método de busca local poderá se utilizar de uma ou várias boas soluções iniciais (alcançadas após a execução do algoritmo genético) e, a partir delas, analisar suas vizinhanças em busca de uma solução que otimize a função objetivo.

Outro trabalho a ser tratado futuramente é a maneira como os operadores de *crossover* realizam a busca no espaço de soluções. Estudar essa característica é fundamental para definir porque um determinado tipo de operador de *crossover* se sobressai a outro para um problema dado. Entender este mecanismo possibilitará a criação de operadores mais versáteis e que encontrem soluções melhores para os problemas abordados.

Referências Bibliográficas

Baker, K. (1974). Introduction to Sequencing and Scheduling, New York.

Campello, R. E. e Maculan, N. (1994). Algoritmos e Heurísticas – Desenvolvimento e Avaliação de Performance, Niterói – Rio de Janeiro, Brasil.

Carvalho, A. C. P. de L. F.; Braga, A. de P.; Ludemir, T. B. (2003). Computação Evolutiva In: Rezende, S. O. *Sistemas Inteligentes – Fundamentos e Aplicações*, São

Paulo, pág. 225 – 248.

Cheng, R.; Gen, M; Tsujimura, Y. (1996). A tutorial survey of job shop scheduling problems using genetic algorithms: representation, *Comput Ind. Eng.*, vol. 30, nº 4, pág. 983 – 997.

Coley, D. A. (1999) An Introduction to Genetic Algorithms for Scientists and Engineers, Singapura.

Costa, A. de O. e Freitas, A. S. (2006) Algoritmos Genéticos: alguns experimentos com os operadores de cruzamento ("crossover") para o Problema do Caixeiro Viajante Assimétrico, *Anais do XXVI Encontro Nacional de Engenharia de Produção.*

Croce, F. della, Tadei, R. e Volta, G. (1995). A Genetic Algorithm for the Job Shop Problem, *Computers Ops Res*. Vol. 22, nº 1, 15 – 24.

Linden, R. (2006). Algoritmos Genéticos – Uma Importante Ferramenta da Inteligência Computacional, Rio de Janeiro – Brasil.

Mattfeld, D. C.; Bierwirth, C. (2004). An Efficient Genetic Algorithm for Job Shop Scheduling with Tardiness Objectives, *European Journal of Operational Research*, vol. 155, pág. 616 – 630.

Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs, Berlin – Germany.

Norvig, P. e Russel, S. (2004). Inteligência Artificial, São Paulo – Brasil.

Reeves, C. R. (org., 1993). Modern Heuristic Search Methods, Orient Longman, Great Britain.