UMA COMPARAÇÃO EMPÍRICA DE OPERADORES DE CROSSOVER PARA O PROBLEMA DE JOB SHOP COM DATAS DE ENTREGAS

Filipe de Oliveira Saraiva (USP)
filipe.saraiva@usp.br
ANTONIO COSTA DE OLIVEIRA (UFPI)
tnycosta@uol.com.br

O problema de programação de tarefas em ambiente job shop com datas de entrega é comumente encontrado na otimização da produção de empresas e indústrias de vários tipos. Nele, busca-se concluir o processamento das tarefas de forma a minimizzar o atraso total em relação às datas de entrega. Devido ao grande número de combinações existentes entre as variáveis de decisão, a obtenção da solução ótima por métodos exatos é normalmente impraticável, dado o alto custo computacional inerente. Frente a este quadro, vários pesquisadores optam por utilizar metaheurísticas, como os algoritmos genéticos, para obter boas soluções ao problema a custo computacional hábil. Este trabalho faz um estudo comparativo entre os principais operadores de crossover existentes na literatura especializada, avaliando o desempenho destes em encontrar boas soluções ao problema de job shop com datas de entrega.

Palavras-chaves: Problema de Job Shop com Datas de Entrega, Algoritmos Genéticos, Operadores de Crossover

1. Introdução

Problemas de otimização combinatória são problemas onde busca-se encontrar um arranjo, grupamento, ordenação ou seleção de coleções de objetos discretos, usualmente finito em número,em que a solução encontrada represente a solução ótima ao problema proposto (CAMPELLO e MACULAN, 1994; REEVES, 1993).

Problemas deste tipo são de fundamental importância para o estudo da otimização de sistemas de produção relacionados tanto a tomada de decisão quanto a alocação de recursos ou sequenciamento, pois os mesmos representam um modelo matemático abstraído de problemas enfrentados por organizações e empresas no mundo real.

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 exatos a um custo computacional razoável (REEVES, 1993).

Visto a importância que o estudo de problemas desse tipo tem para a otimização de sistemas de produção em diferentes modelos de organizações atuais, e o alto custo computacional inerente a resolução exata desses problemas, alguns pesquisadores optaram por uma abordagem heurística a este desafio.

Heurísticas são métodos de resolução de problemas de otimização que buscam uma boa solução (não necessariamente a ótima), aceitável para o modelo do problema abordado, a um custo computacional razoável (REEVES, 1993).

Assim como as heurísticas, metaheurísticas bem projetadas conseguem encontrar boas soluções a um custo computacional viável. A diferença entre elas encontra-se nas estratégias implementadas pelas metaheurísticas, que objetivam escapar de ótimos locais e continuar a busca pelo espaço de soluções (GLOVER e KOCHENBERGER, 2003).

Este trabalho trata sobre a metaheurística algoritmo genético, realizando um estudo comparativo entre os principais operadores genéticos de *crossover* presentes na literatura, aplicados ao problema de *job shop* com datas de entrega, caracterizado como um problema de otimização combinatória do tipo NP-Árduo (GAREY, JOHNSON e SETHI, 1976).

Na próxima seção, descreve-se o problema de *job shop* em si, suas características e sua variação com acréscimo de datas de entrega; a Seção 3 discorre sobre o conceito de algoritmos genéticos; a Seção 4 descreve as características do algoritmo genético implementado, assim como os tipos de operadores de *crossover* aplicados e seus fundamentos. Finalmente, a Seção 5 descreve as instâncias dos problemas avaliados e mostra os resultados obtidos. Em seguida, a Seção 6 expõe as conclusões obtidas e pesquisas futuras e, nos anexos, apresentamos as tabelas de execução do algoritmo para as instâncias propostas.

2. O Problema de *Job Shop* com Datas de Entrega

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 *k*-operações. 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 alguma medida de desempenho tais como: *makespan* (concluir todas as tarefas no menor tempo possível), atraso total (concluir todas as tarefas com o menor atraso total em relação as suas datas de entrega), tempo médio de fluxo, número de tarefas atrasadas, etc. (BAKER, 1998).

O atraso T_i de uma tarefa j_i é calculado através da seguinte fórmula:

$$T_i = max\{(c_i - d_i), 0\}$$

onde c_i e d_i representam o instante de conclusão e a data de entrega da tarefa j_i, respectivamente. O Atraso Total é calculado por:

$$T = \sum_{i=1}^{n} T_i$$

O problema de *job shop* com datas de entrega consiste em programar as operações das tarefas de forma a minimizar o atraso total.

3. Algoritmos Genéticos

Algoritmos genéticos estão entre as metaheurísticas mais estudadas e desenvolvidas nas últimas décadas. Criada por John Holland na década de 70, baseia-se na dinâmica natural das espécies para fazer evoluir um conjunto de soluções para um determinado problema, afim de, a cada geração, as soluções encontradas vão sendo melhor adaptadas – tem um desempenho melhor na função objetivo – para o ambiente em que se encontra – as restrições do problema. Sua concepção lembra bastante a teoria darwinista de evolução (COLEY, 1999).

A representação de soluções em um algoritmo genético é feita através de uma *string* de caracteres, chamada de cromossomo, onde cada caracter é comumente chamado de gene. Existem vários tipos de codificação, dentre os quais podemos destacar: codificação binária, ponto flutuante, inteiros, etc. Esse tipo de modelagem facilita os processos de cruzamento (*crossover*) e mutação, principais operadores no processo de evolução desenvolvido pelo método (COLEY, 1999).

A codificação indica de que maneira uma solução será representada, e isso permite executar a busca no espaço codificado de soluções.

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 caracteres 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, formando indivíduos melhores adaptados ao ambiente – ou seja, que tenham um melhor desempenho na função objetivo (CARVALHO, BRAGA e LUDEMIR, 2003). Existem vários tipos de operadores de *crossover* que são aplicados com êxito a problemas de otimização combinatória. Dentre estes, podemos citar os *crossovers PMX*, *OX*, *CX*, *LOX*, e *PPX*, bastante empregados em algoritmos genéticos com codifição inteira. O conceito de cada um será apresentado adiante.

Em seguida inicia-se o processo de mutação. Este operador serve para inserir na população aqueles genes que, porventura, se perderam durante o processo de cruzamento e formação de gerações, tornando a busca mais diversificada. Normalmente, utilizam-se baixos valores para este operador, variando entre 0.01 e 0.05 (COLEY, 1999).

A partir dessas etapas, tem-se novas soluções que farão parte da próxima geração do genético, repetindo o ciclo até ser alcançado um número máximo de gerações pré-estabelecido ou outro critério de parada, como tempo máximo de processamento de execução do método.

4. Algoritmo Genético para o Problema de *Job Shop* com Datas de Entrega

No algoritmo genético implementado, utilizou-se a representação inteira codificada em lista de preferência (CHENG, GEN e TSUJIMURA, 1996). O critério de seleção utiliza o método da roleta (COLEY, 1999).

Optou-se pelo uso do operador de inversão como forma de mutação (CROCE, TADEI e VOLTA, 1995). Neste operador, selecionam-se dois genes do cromossomo e inverte-se a subcadeia de genes compreendidos entre estes dois pontos. A Figura 1 traz um exemplo.

Indivíduo: (2 3 4 1 0)

(2 |3 4 1| 0)

Resultado: (2 1 4 3 0)

Figura 1 – Exemplo de inversão

O critério de parada utilizado no presente trabalho foi o número máximo de gerações no valor de 1000 e 10000 iterações. O motivo era investigar se, com um aumento substancial no valor desse parâmetro, o algoritmo encontraria soluções de qualidade muito diferentes entre si.

As outras características do algoritmo genético implementado foram: valor da taxa de mutação em 0.01; taxa de *crossover* em 0.9. A população inicial é gerada de forma aleatória, e o melhor indivíduo de cada geração é mantido 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* tem por finalidade possibilitar que bons genes da população possam se combinar, formando indivíduos com melhor desempenho na função objetivo (CARVALHO, BRAGA e LUDEMIR, 2003). Neste trabalho implementaram-se os *crossovers PMX*, *OX*, *CX*, *LOX*, e *PPX*, retirados da literatura onde os mesmos foram aplicados com resultados satisfatórios a vários problemas de otimização combinatória.

4.1 *PMX – Partially Mapped Crossover*

O operador *PMX* foi proposto por Goldberg e Lingle (1985 apud POTVIN, 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 (COSTA e ANDERSON, 2006). Tomemos a Figura 2 como exemplo:

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

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

Figura 2 – Exemplo de aplicação do *crossover PMX*

No caso acima, os cortes em *p1* (que cobrem os genes 3, 4 e 6) e os cortes em *p2* (nos genes 1, 4 e 2) criam o relacionamento 3 ↔ 1, 4 ↔ 4 e 6 ↔ 2. Esse mapeamento significa que, após a troca de genes pelo *PMX*, os genes fora da subcadeia deverão ser mudados seguindo estas regras afim de tratar a inviabilidade. Por exemplo, o gene 3 em *f1* herdado de *p2* terá que ser mudado para 1, de acordo com o mapeamento.

4.2 *OX – Order Crossover*

O operador *OX* foi proposto por Davis (1985 apud MICHALEWICZ, 1996) também para o Problema do Caixeiro Viajante. Assim como o *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 (COSTA e ANDERSON, 2006). A Figura 3 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 3 – Exemplo de aplicação do *crossover OX*

4.3 *CX – Cycle Crossover*

O operador *CX* foi proposto por Oliver (1987 apud POTVIN, 1996) e aplicado ao Problema do Caixeiro Viajante. Ele trabalha sobre um subconjunto de genes que ocupam um mesmo conjunto ou formam um "ciclo" 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 4 apresenta um exemplo:

p1: (**1** 2 3 **4 6 5)** → *f1*: (1 3 2 4 5 6)

p2: (**5** 3 2 **1 4 6)** → *f2*: (5 2 3 1 4 6)

Figura 4 – Exemplo de aplicação do *crossover CX*

4.4 *LOX – Linear Order Crossover*

O operador *LOX* é uma variação do operador *OX* e foi desenvolvido por Falkenauer e Bouffouix (FALKENAUER e BOUFFOUIX, 1991). 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 a Figura 5:

p1: (1 2 |3 4| 6 5) → (h h 3 4 6 5) → (3 4 h h 6 5) → *f1*: (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) → *f2*: (5 2 3 4 1 6)

Figura 5 – Exemplo de aplicação do *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 6, 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 6 – Exemplo de aplicação do *crossover PPX*

Com os operadores de *crossover* apresentados, a próxima seção discorrerá sobre o conjunto de testes e os resultados obtidos.

5. Resultados Computacionais

O conjunto das instâncias utilizadas para realizar os testes foram retirados dos problemas de *job shop* da OR-Library (BEASLEY, 1990). Estas instâncias foram originalmente desenvolvidas para testes com o objetivo de otimizar o *makespan.* Devido a isso, foram utilizadas as seguintes expressões afim de gerar valores de datas de entrega para cada tarefa:

$$d_j = b * \sum_{i=1}^{m} t_{ij}$$

$$b = ((J * M)/1000) + 0{,}5$$

onde J representa o número de tarefas, M o número total de máquinas, t_{ij} o tempo de processamento da cada operação i referente a tarefa j e d_j a data de entrega para cada tarefa j.

As instâncias utilizadas tem o número de operações igual ao número de máquinas. A Tabela 1 apresenta as instâncias do problema e seus respectivos números de tarefas e máquinas:

Instâncias	Nº de Tarefas	Nº de Máquinas
ft06	6	6
abz5	10	10

abz8	20	15
yn1	20	20
swv20	50	10

Tabela 1 – Características das instâncias dos problemas utilizados

Cada algoritmo genético foi executado 10 vezes para cada instância e para cada critério de parada, obtendo-se a média e o melhor resultado destas execuções. Tanto as melhores médias quanto os melhores resultados obtidos estão destacados em negrito nas tabelas. A Tabela 2 apresenta a média e os melhores resultados de cada algoritmo abordando a instância ft06; a Tabela 3 para a instância abz5; Tabela 4 para abz8, Tabela 5 para yn1 e Tabela 6 para swv20. Os gráficos representam os valores das médias para ambos os critérios de parada utilizados.

	ft06			
	1000 Gerações		**10000 Gerações**	
Crossover	**Média**	**Melhor**	**Média**	**Melhor**
PMX	194,9	171,5	181,9	**166,5**
OX	198,9	191,5	187,6	179,5
LOX	190,9	170,5	**174,6**	**166,5**
CX	**187,9**	**169,5**	178,3	**166,5**
PPX	194,6	182,5	183,3	171,5

Tabela 2 – Resultados para a instância ft06

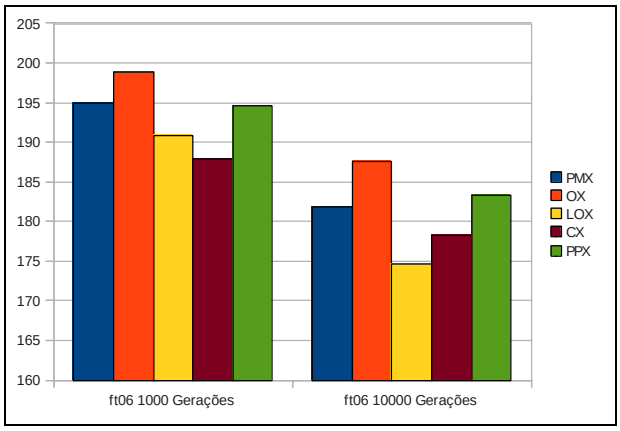

Figura 7 – Gráficos para o resultado da média na instância ft06

	abz5			
	1000 Gerações		**10000 Gerações**	
Crossover	**Média**	**Melhor**	**Média**	**Melhor**
PMX	12330,9	11329,5	11740,4	10840,5
OX	11996,1	11305,5	11179,9	10590,5
LOX	12454	11851,5	11906,4	11409,5
CX	**9715,3**	**8800,5**	**9193,3**	**8006,5**
PPX	12301,8	11648,5	11485,2	10717,5

Tabela 3 – Resultados para a instância abz5

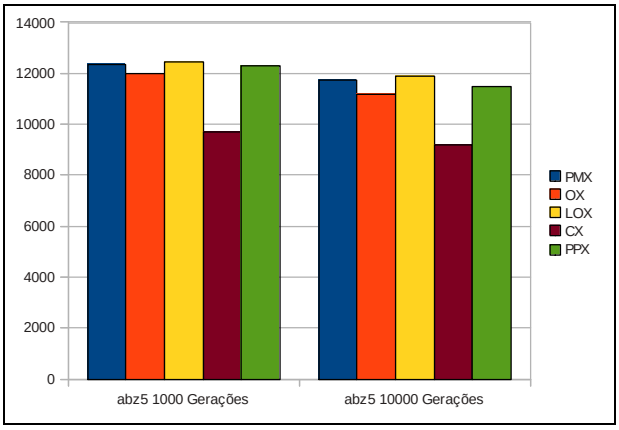

Figura 8 – Gráficos para o resultado da média na instância abz5

	abz8			
	1000 Gerações		10000 Gerações	
Crossover	**Média**	**Melhor**	**Média**	**Melhor**
PMX	26225,2	24851	24500,9	21190
OX	26025,2	25378	24309,8	22357
LOX	26010	24731	24839,4	23849
CX	**19202,2**	**18138**	**15607,9**	**14550**
PPX	26316,9	25257	24672,8	23170

Tabela 4 – Resultados para a instância abz8

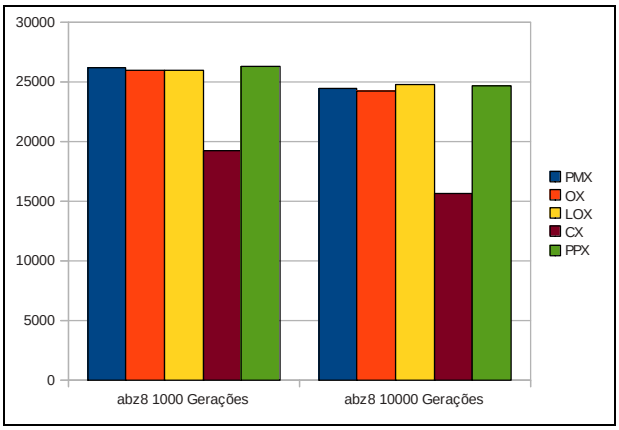

Figura 9 – Gráficos para o resultado da média na instância abz8

	yn1			
	1000 Gerações		10000 Gerações	
Crossover	**Média**	**Melhor**	**Média**	**Melhor**
PMX	42010,6	39157	39776	37244
OX	41501,1	39869	38940,9	36697
LOX	42536,2	40667	40183,4	37914
CX	**30068,2**	**27146**	**23679,1**	**20355**
PPX	42097,3	39938	40140,5	37408

Tabela 5 – Resultados para a instância yn1

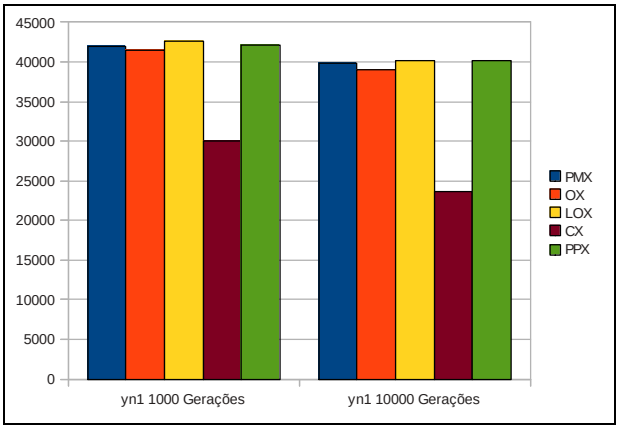

Figura 10 – Gráficos para o resultado da média na instância yn1

	swv20			
	1000 Gerações		10000 Gerações	
Crossover	**Média**	**Melhor**	**Média**	**Melhor**
PMX	247410	241780	239245,2	235692
OX	242746,4	215742	234212,6	222712
LOX	245602,2	242578	237274,4	233000
CX	**184568**	**173974**	**149399,6**	**139500**
PPX	248033	243354	239743	232012

Tabela 6 – Resultados para a instância swv20

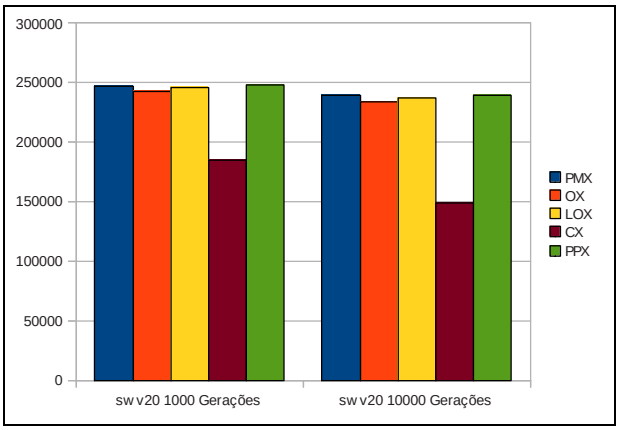

Figura 11 – Gráficos para o resultado da média na instância swv20

Pelos resultados obtidos, percebe-se que o operador *CX* foi o que conseguiu o melhor desempenho para a maioria das instâncias propostas entre os operadores de *crossover* avaliados, exceto para a instância *ft06* com critério de parada em 10000 gerações, onde o operador *LOX* obteve uma média ligeiramente melhor. Entretanto, o operador *CX* conseguiu atingir o melhor resultado para esta instância, assim como os operadores *PMX* e o próprio *LOX*.

Abaixo temos a Figura 12 com uma comparação do desempenho dos resultados médios do atraso total para os dois critérios de parada utilizados. Para cada instância, o conjunto de

pontos à esquerda representa o resultado para o critério de parada em 1000 gerações; o conjunto de pontos mais a direita representa o valor obtido para esta mesma instância com o critério de parada em 10000 gerações.

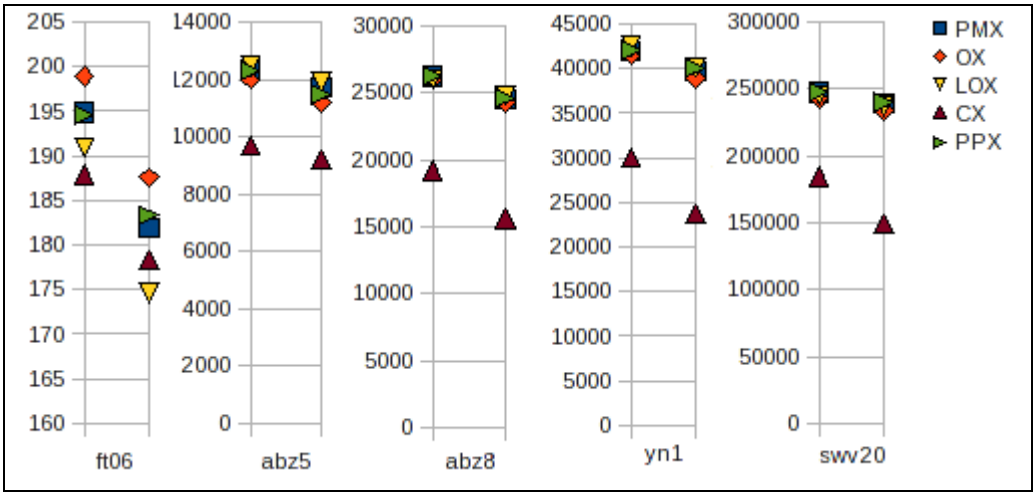

Figura 12 – Comparação das médias entre os métodos

A Figura 13 abaixo, apresenta um gráfico comparativo entre as melhores soluções encontradas em nossas execuções para as instâncias do problema aqui trabalhado. Novamente, os pontos à esquerda em cada gráfico representam os resultados para o critério de parada em 1000 gerações; o conjunto de pontos mais a direita representa o valor obtido para esta mesma instância com o critério de parada em 10000 gerações.

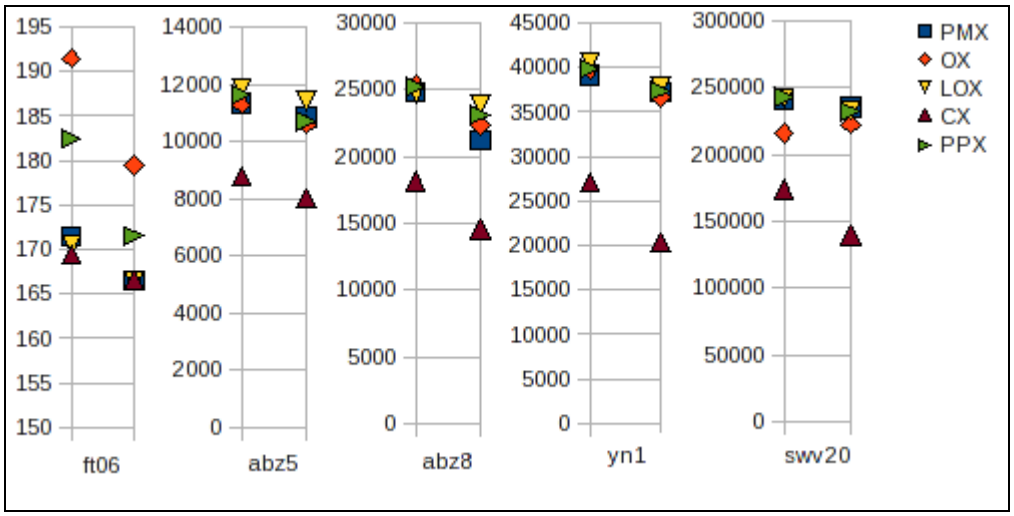

Figura 13: Comparação dos melhores resultados entre os métodos

Pelos gráfico, percebe-se o bom desempenho do operador *CX* em encontrar soluções de boa qualidade, tanto quando comparamos as médias quanto quando comparamos as melhores soluções encontradas para ambos os critérios de parada utilizados nesta pesquisa.

6. Conclusão e Trabalhos Futuros

Este trabalho fez um estudo comparativo entre os operadores de *crossover PMX, OX, CX, LOX,* e *PPX*, aplicados ao problema de *job shop* com datas de entrega.

Estes operadores estão presentes na literatura para vários tipos de problemas de otimização combinatória. Implementados para o problema ao qual este artigo trata, percebe-se que o operador *CX*, em geral, teve um desempenho melhor que o dos outros operadores, apesar de seu resultado médio para o critério de parada em 10000 gerações na instância *ft06* ter sido ligeiramente inferior ao do operador *LOX*.

Percebemos que essa característica do operador *CX* está relacionada com a forma com que o operador faz a busca no espaço de soluções. Ao contrário dos demais operadores, para a codificação utilizada, existe apenas uma forma de realizar o cruzamento com *CX* – ou seja, sua vizinhança é composta de apenas duas outras soluções.

Mesmo elevando o número de iterações, o operador *CX* ainda fez uma busca com resultados melhores que os demais operadores, exceto para a instância *ft06* em 10.000 gerações. Isso significa que os outros tipos de *crossover* "se perdem" no espaço de busca, não conseguindo realizar uma intensificação em um subespaço de forma eficiente. Ao contrário, o aumento no número de iterações proporcionou ao operador *CX* melhorar o desempenho de sua busca em relação aos outros operadores, otimizando tanto o resultado das médias quanto o resultado do melhor valor encontrado, em uma proporção maior que os demais tipos de *crossover* aqui comparados.

O aumento no número de gerações fez com que o operador *CX* intensifica-se a busca em uma subregião do espaço de busca que, supostamente, se mostrou atrativa durante a execução do método. Essa característica não foi verificada nos demais operadores de *crossover* aqui estudados, o que nos leva a crer que os mesmos não conseguem convergir em um subespaço de busca promissor quanto à qualidade das soluções.

Ao perceber que mesmo aumentando em 10 vezes o número de gerações os operadores de *crossover PMX*, *OX*, *LOX* e *PPX* não conseguiram fazer o algoritmo genético convergir a uma sub-região do espaço de busca, podemos concluir que é muito provável que os mesmos não realizam uma busca eficiente quanto à intensificação utilizando a codificação aqui implementada.

Esperamos continuar com este trabalho, utilizando agora outras representações de cromossomo. Investigaremos a possibilidade dessa configuração do comportamento dos operadores genéticos ocorrer em outras condições, como por exemplo, quando usamos outro tipo de codificação. Outro trabalho futuro é pesquisarmos quais são as características que tornam um operador de *crossover* mais apto a fazer uma busca e encontrar boas soluções, em comparação a outro. Para este novo trabalho, far-se-á necessário entender os mecanismos de busca inerentes a cada tipo de *crossover*, e como esse mecanismo reflete na função de vizinhança e na criação do espaço de busca codificado.

Os estudos comparativos e pesquisas na área de metaheurísticas conduz ao desenvolvimento de métodos de busca cada vez mais robustos para vários problemas. As comparações entre métodos e operadores são de fundamental importância para o desenvolvimento da área.

Referências

Baker, K. *Elements of Sequencing and Scheduling*. Hanover - NH, 1998.

Beasley, J.E. *OR-Library: distributing test problems by electronic mail*. Journal of the Operational Research Society 41(11), p.1069-1072, 1990.

Campello, R. E. e Maculan, N. *Algoritmos e Heurísticas – Desenvolvimento e Avaliação de Performance*. Rio de Janeiro: Ed. da Universidade Federal Fluminense, 1994.

Carvalho, A. C. P. de L. F.; Braga, A. de P.; Ludemir, T. B. *Computação Evolutiva* In: Rezende, S. O.

Sistemas Inteligentes – Fundamentos e Aplicações. São Paulo: Ed. Manole, p.225 – 248, 2003.

Cheng, R.; Gen, M; Tsujimura, Y. *A tutorial survey of job shop scheduling problems using genetic algorithms I - representation.* Computers & Industrial Engineering., vol. 30, no 4, p. 983 – 997, 1996.

Coley, D A. *Introduction to Genetic Algorithms for Scientists and Engineers*. Singapore: World Scientific, 1999.

Costa, A. de O. e Freitas, A. S. *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, 2006.

Croce, F. della, Tadei, R. e Volta, G. *A Genetic Algorithm for the Job Shop Problem.* Computers & Operations Research., vol. 22, no 1, p.15 – 24, 1995.

Falkenauer, E. e Bouffouix, S. *A genetic algorithm for job shop*. Proceedings of the IEEE International Conference on Robotics and Automation , 1991.

Garey, M. R., Johnson, D. S. e Sethi, R. *The complexity of flowshop and job-shop scheduling.* Mathematics of Operations Research, vol. 1, no 2, p.117 – 129, 1976.

Glover, F. e Kochenberger, G. A. *Handbook of Metaheuristics*. Massachusetts: Kluwer Academic, 2004.

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

Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, Germany: Springer, 1996.

Potvin, J. Y. *Genetic Algorithms for the Traveling Salesman Problem*. Annals of Operations Research 63, p.339 – 370, 1996.

Reeves, C. R. *Modern Heuristic Techniques for Combinatorial Problems*. Oxford: Blackwell Scientific Press, 1993.

ANEXOS

Tabela com os resultados das 10 execuções do algoritmo genético para cada um dos tipos de *crossover* implementados, para 1.000 gerações.

1.000Gerações													
Problemas			Execuções										
Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
ft06	6	6	PMX	204,5	193,5	191,5	201,5	204,5	190,5	199,5	171,5	197,5	194,5	194,9
			OX	198,5	191,5	192,5	197,5	197,5	215,5	207,5	192,5	200,5	195,5	198,9
			LOX	190,5	184,5	170,5	202,5	186,5	197,5	200,5	198,5	171,5	206,5	190,9
			CX	192,5	185,5	171,5	192,5	205,5	195,5	184,5	169,5	193,5	188,5	187,9
			PPX	191,5	202,5	182,5	202,5	186,5	191,5	194,5	202,5	188,5	203,5	194,6
Problemas			Execuções											
Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
abz5	10	10	PMX	11578,5	12994,5	12326,5	12097,5	12041,5	12793,5	12883,5	13007,5	12256,5	11329,5	12330,9
			OX	11540,5	12458,5	11305,5	12283,5	12319,5	11677,5	11753,5	11490,5	12027,5	13104,5	11906,1
			LOX	12520,5	11879,5	13276,5	12755,5	12683,5	11851,5	12600,5	12114,5	12722,5	12135,5	12454
			CX	8898,5	9930,5	10011,5	10401,5	11939,5	9372,5	8936,5	9989,5	8872,5	8800,5	9715,3
			PPX	11870,5	12498,5	12684,5	12264,5	12671,5	12314,5	11861,5	11648,5	12957,5	12246,5	12301,8
Problemas			Execuções											
Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
abz8	20	15	PMX	26680	26769	26837	25144	24851	26320	26891	26312	26001	26447	26025,2
			OX	27421	25599	26149	25655	25378	25880	26223	25721	26272	25954	26025,2
			LOX	25896	26125	26764	26868	26713	26072	24731	26621	24881	25429	26010
			CX	18717	19867	18807	19682	19473	19417	18138	18268	18226	21427	19202,2
			PPX	25257	26360	26211	25435	25781	26758	27240	26609	26669	26849	26316,9
Problemas			Execuções											
Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
yn1	20	20	PMX	42039	43860	42419	39157	39717	42488	42854	43122	42141	42309	42010,6
			OX	42479	43272	39947	41525	42927	40383	39954	41161	43494	39869	41501,1
			LOX	42523	42892	40667	43604	41891	42850	42376	42469	42628	43462	42536,2
			CX	27146	31900	30244	32299	29093	27871	30913	30004	29377	31835	30068,2
			PPX	43214	42584	40929	39938	40914	44245	42485	42821	41891	41952	42097,3
Problemas			Execuções											
Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
swv20	50	10	PMX	241780	253024	248866	247474	244056	246928	248384	247532	251370	244686	247410
			OX	252070	215742	247726	249798	235242	249852	240530	242396	239206	254902	242746,4
			LOX	244394	243122	243352	246122	254078	242578	244850	248080	245404	244042	245602,2
			CX	185876	186428	182034	177640	173974	189060	182162	188662	184166	195678	184568
			PPX	250666	254094	244318	243354	248224	243906	250760	248456	253138	243414	248033

Tabela com os resultados das 10 execuções do algoritmo genético para cada um dos tipos de *crossover* implementados para 10.000 gerações.

10.000 Gerações

Problemas	Número de Jobs	Número de Máquinas	Crossover	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	6ª Execução	7ª Execução	8ª Execução	9ª Execução	10ª Execução	Média
				Problemas					Execuções					
ft06	6	6	PMX	184,5	193,5	166,5	190,5	190,5	191,5	180,5	166,5	171,5	183,5	181,9
			OX	186,5	191,5	193,5	184,5	185,5	194,5	182,5	179,5	185,5	192,5	187,6
			LOX	166,5	184,5	170,5	171,5	184,5	181,5	166,5	182,5	166,5	171,5	174,6
			CX	166,5	166,5	185,5	182,5	185,5	182,5	166,5	181,5	182,5	183,5	178,3
			PPX	182,5	182,5	183,5	186,5	187,5	171,5	183,5	189,5	184,5	181,5	183,3
				Problemas					Execuções					
abz5	10	10	PMX	12024,5	11841,5	12044,5	10840,5	11650,5	11661,5	11413,5	11898,5	12172,5	11856,5	11740,4
			OX	11036,5	11987,5	10905,5	10819,5	10590,5	10844,5	11533,5	11382,5	11509,5	11189,5	11179,9
			LOX	11409,5	11791,5	11797,5	12137,5	11837,5	11844,5	12021,5	12031,5	12028,5	12164,5	11906,4
			CX	9730,5	8752,5	10561,5	9133,5	8006,5	8796,5	9423,5	8610,5	9815,5	9102,5	9193,3
			PPX	11397,5	11775,5	11389,5	11535,5	11810,5	11638,5	11348,5	11240,5	10717,5	11998,5	11485,2
				Problemas					Execuções					
abz8	20	15	PMX	25245	24520	21190	24805	24628	25513	24544	24439	25729	24396	24500,9
			OX	24217	22357	23877	25044	24737	24864	23760	24550	25706	23986	24309,8
			LOX	23952	24394	25260	24624	25486	25878	23849	24881	26042	24028	24839,4
			CX	15748	15750	16575	14826	14655	15130	16086	17489	15270	14550	15607,9
			PPX	23170	25565	25180	23173	25537	25034	25055	25439	23665	24910	24672,8
				Problemas					Execuções					
yn1	20	20	PMX	40338	40301	38721	37244	41103	39497	41242	39681	40227	39406	39776
			OX	38642	40131	40511	39662	36697	40184	37323	38869	39483	37907	38940,9
			LOX	40317	40190	37914	40410	41359	41198	39730	40310	40316	40090	40183,4
			CX	24224	24542	20612	27256	22794	25610	22625	20355	25362	23411	23679,1
			PPX	40896	40955	40348	39452	40451	40742	40334	39921	37408	40898	40140,5
				Problemas					Execuções					
swv20	50	10	PMX	241504	238504	235816	235692	238528	240102	242530	242026	241278	236472	239245,2
			OX	240452	234856	228276	231864	240410	235262	229066	237076	222712	242152	234212,6
			LOX	233114	241436	233204	238140	238794	236634	241626	233000	241062	235734	237274,4
			CX	150802	151446	162364	139500	140834	147046	147454	154384	155520	144646	149399,6
			PPX	232012	239000	242022	242148	240248	240778	243724	234970	243206	239322	239743