ALGORITMO GENÉTICO PARA PROBLEMAS DE *JOB SHOP*

Fabrício Vivas Andrade

vivas@dcc.ufmg.br

Raoni Maira Resende

raoni@dcc.ufmg.br

RESUMO

O objetivo deste trabalho é implementar um algoritmo genético para solucionar problemas de sequenciamento, especialmente, o problema de manufatura do tipo Job Shop.

1. INTRODUÇÃO

Os problemas de sequenciamento são problemas de alocação de recursos no tempo para se executar um conjunto de tarefas que constituem um processo ou um produto [BZ96]. Os problemas de sequenciamento mais conhecidos se encontram nas áreas de manufatura e computação.

Em problemas desse tipo, as tarefas competem por recursos que podem ser de diferentes naturezas como, por exemplo, processadores, máquinas em uma linha de produção, horários e dinheiro.

Os problemas da manufatura podem ser enunciados da seguinte forma:

Dados um conjunto de tarefas (operações) que constituem os n produtos e um conjunto de m recursos (máquinas), pergunta-se: como e quando alocar as máquinas para realização das tarefas de modo a atender algum critério de otimização?

Os principais problemas de sequenciamento na manufatura são conhecidos como *Flow Shop, Open Shop* e *Job Shop*.

Flow Shop é um tipo de problema na manufatura em que todos os produtos, também chamados de *jobs,* passam por todas as máquinas na mesma ordem, isto é, possuem a mesma seqüência tecnológica. *Open Shop* é um problema onde a sequência tecnológica do produto não importa. E, finalmente, o problema é conhecido como *Job Shop* se cada produto possui sua própria sequência tecnológica de fabricação.

O problema de *Job Shop* é a forma mais geral de problemas de manufatura e por isso foi escolhido como objeto de estudo desse trabalho.

Dentre os critérios de otimização, o de minimização do tempo de término de todas as tarefas, também chamado de *makespan,* é o mais estudado. Esse é um problema clássico de otimização combinatória e é considerado um dos problemas mais intratáveis computacionalmente[LL93].

Instâncias bem particulares do problema com apenas duas máquinas e um número pequeno de *jobs* podem ser resolvidos eficientemente. Mas problemas maiores, com pelo menos três máquinas e três *jobs*, são NP-difícil [GS78]. Para um problema do tipo *Job Shop* completo, ou seja, que todos os *jobs* passam por todas as máquinas o número de sequenciamentos possíveis é igual a $(n!)^m$ onde n é o número de *jobs* e m é o número de máquinas.

2. MOTIVAÇÃO E MÉTODOS DE RESOLUÇÃO

A principal motivação desse trabalho vem do fato que problemas de *Job Shop* são muito recorrentes na manufatura. Além disso, grande parte das indústrias ainda não utiliza métodos computacionais para resolvê-lo, baseando-se apenas na experiência do gerente de produção da empresa.

Como o problema de *Job Shop* já vem sendo estudado a mais de 30 anos, diferentes abordagens já foram propostas. O *Branch and Bound,* para soluções exatas e o *Shifting Bottleneck* para soluções aproximadas são duas abordagem muito usadas.

Algoritmos Genéticos fornecem bons resultados principalmente para instâncias de difícil

resolução com métodos aproximados como visto em[NY91], [CTV95] e [CV95].

A abordagem desse artigo consiste em propor uma nova implementação usando Algoritmos Genético para resolver o problema. A principal diferença entre a proposta atual em relação a anteriores [CTV95] e [CV95], consiste na definição de novos operadores e definição simplificada do cromossomo como será descrito na seção 4.

Abordagens para o problema utilizando outros métodos heurísticos, como *Simulated Anealing* [LAL92] e *Tabu-Search* [TA93], também podem ser encontrados na literatura, porém não são objeto de estudo desse trabalho.

3. O PROBLEMA DE JOB SHOP

3.1. Definição Formal

O problema de *Job Shop* pode ser definido formalmente como um problema de minimização da seguinte maneira:

$$\text{Min } t_n$$
$$t_j - t_i \geq p_i \qquad \forall (T_i, T_j) \in \text{A}$$
$$t_i - t_j \geq p_j \text{ ou } t_j - t_i \geq p_i \; \forall (T_i, T_j) \in E_k \forall P_k \in \text{P}$$
$$t_i \geq 0 \qquad \forall T_i \in \text{T}$$

Onde t_n é o *makespan*, t_i e t_j são o tempo de início da tarefa T_i ou T_j e p_i representa o tempo de processamento da tarefa i

$\text{T} = \{T_0, T_1, T_2, T_3, \cdots, T_n\}$ representa o conjunto de tarefas que devem ser executadas, sendo que T_0 representa a fonte e T_n o sumidouro, que são tarefas criadas para representar as tarefas inicial e final do sequenciamento. P representa o conjunto de *m* máquinas e A corresponde ao conjunto de pares ordenado (T_i, T_j) em que existe uma relação de precedência $T_i \prec T_j$. E_k é o conjunto de pares de tarefas que são executadas na mesma máquina.

A primeira restrição garante que a seqüência de processamento das tarefas para cada *job* será respeitada. Já a segunda restrição garante que uma máquina só executará uma tarefa por vez. Assim, qualquer solução que respeite as restrições é uma solução viável.

Uma boa representação por grafo disjuntivo para este problema foi proposta em [RS64] e será descrita na seção a seguir.

3.2. Representação por grafo disjuntivo

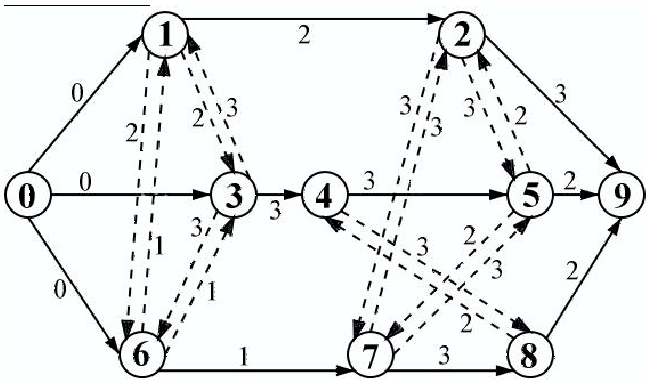
Uma forma de melhorar o entendimento e a visualização do problema é representá-lo por um grafo disjuntivo. Cada tarefa é representada por um vértice e os arcos indicam a precedência das mesmas. Para cada duas tarefas consecutivas do mesmo *job* existe um arco direto (i, j) e para cada par de tarefas $(T_i, T_j) \in E_k$ executadas na mesma máquina existem dois arcos (i, j) e (j, i) com sentidos opostos. O peso de cada arco é o tempo gasto pela tarefa origem do arco.

Os arcos que conectam as tarefas de um mesmo *job* são denominados arcos conjuntivos e os arcos conectando tarefas de uma mesma máquina são denominados arcos disjuntivos. Podemos observar, de acordo com a definição, que os arcos disjuntivos formam um clique para cada máquina.

Essa representação é exemplificada pelo problema a seguir, composto por três *jobs* e três máquinas.

Tabela 1

Produto	*tarefa:maq (tempo de processamento)*		
1	1:1(2)	2:2(3)	
2	3:1(3)	4:3(3)	5:2(2)
3	6:1(1)	7:2(3)	8:3(2)



Encontrar uma solução válida nesse grafo disjuntivo consiste em selecionar apenas um dos

arcos que conectam as tarefas $(T_i, T_j) \in E_k$ para todas as máquinas, ou seja, selecionar um sentido para cada par escolhendo-se (i, j) ou (j, i). Além disso, o grafo resultante não pode possuir ciclos, para respeitar a ordem de precedência das tarefas.

O *makespan* da solução é o caminho crítico do grafo que é encontrado calculando-se o caminho mais longo no grafo.

O algoritmo de caminho mais longo foi obtido a partir de uma simples alteração do algoritmo de caminho mais curto obtido em [ZIV04].

4. ALGORITMO GENÉTICO

Algoritmos genéticos são algoritmos de buscas baseados no mecanismo de seleção natural e genética. Um conjunto de soluções é criado aleatoriamente, a partir daí, soluções são combinadas por meio de operações naturais (cruzamento e mutação) criando novos indivíduos, também chamados de cromossomos. A busca é guiada para melhores soluções, pois o mecanismo de seleção natural descarta as soluções menos adaptadas e mantêm as melhores.

Uma boa introdução ao tema pode ser encontrada em [GB89].

4.1. Codificação dos Indivíduos

Segundo [MZ96], as principais codificações utilizadas para indivíduos são a real (*float*) e a binária. Para problemas de otimização de funções contínuas utiliza-se a codificação binária. No caso de problemas de *Job Shop*, a utilização de uma codificação binária não é interessante, uma vez que existe a necessidade de verificar a todo o momento se os novos indivíduos criados são viáveis.

A codificação real oferece grandes vantagens tanto por parte de viabilidade dos novos indivíduos quanto na própria implementação dos operadores.

Para se definir o formato dos cromossomos, precisamos entender melhor como são dadas as soluções para um problema de *Job Shop*. Para isso utilizaremos o problema descrito pela tabela a seguir.

Tabela 2

Produto	*tarefa:maq (tempo de processamento)*		
1	1:1(2)	2:2(3)	
2	3:1(3)	4:3(3)	5:2(2)
3	6:1(1)	7:2(3)	8:4(2)
4	9:1(4)	10:4(1)	11:3(2)
5	12:4(4)	13:3(4)	

Pode-se ver pela tabela 2 que as tarefas 1,3,6,9 são executadas na máquina 1. Se descobrirmos a ordem das tarefas nessa máquina e nas subsequentes, teremos o sequenciamento final (veja seção 3.2). Em suma, se sequenciarmos as máquinas, estaremos definindo todo o sequenciamento.

Uma solução ótima (*makespan* igual a 13) do problema no exemplo é dado pelo seguinte sequenciamento de tarefas em cada uma das máquinas:

Maq. 1: 3, 6, 9, 1.

Maq. 2: 7, 5, 2.

Maq. 3: 4, 13, 11.

Maq. 4: 12, 10, 8.

O sequenciamento em uma máquina é denominado subcromossomo. Portanto, temos para o problema acima 4 subcromossomos, pois existem 4 máquinas. A união de todos os subcromossomos forma um indivíduo que pode ser uma solução do problema.

Exemplo:

M1 M2 M3 M4

| 3 9 1 6 | 5 2 7 | 11 13 4 | 12 10 8 |

Esta seria uma solução para o problema, observe que entre as barras existem os sequenciamentos das máquinas (subcromossomos) de M1 a M4. É importante notar que um indivíduo do tipo | 2 9 1 6 | 5 3 7 | 11 13 4 | 12 10 8 | não seria uma solução do problema, pois a tarefa 2 e a tarefa 3 não pertencem as máquinas 1 e 2

4.2. Operadores

Definido o formato dos indivíduos podemos escolher os operadores. Em algoritmos genéticos pode-se trabalhar com dois tipos de operadores: livres ou restritos. Operadores restritos somente criam soluções viáveis. Operadores livres criam tanto soluções viáveis quanto inviáveis.

A implementação dos operadores proposta em [CTV95] gera apenas soluções viáveis, mas é

muito complexa e pouco detalhada. Assim, optou-se trabalhar com operadores livres, detectando e eliminando indivíduos inviáveis.

Os operadores utilizados serão apresentados a seguir e foram baseados nos descritos em [CV95] e [CTV95] com algumas alterações.

4.2.1. (Per) Mutação

Para o tipo de codificação escolhida, a mutação não funciona como em binário. Não podemos apenas mudar o número de uma tarefa em um indivíduo porque estaríamos gerando um novo indivíduo fora do espaço de soluções. Exemplo:

C1 | 3 9 1 6 | 5 2 7 | 11 13 4 | 12 10 8 |
* (gene de mutação)

mutação

C1* | 3 7 1 6 | 5 2 7 | 11 13 4 | 12 10 8 |
*

O individuo C1* é um indivíduo inviável porque insere uma tarefa que já existia e ainda coloca a tarefa 7 da M2 em M1.

Com o objetivo de só criar indivíduos viáveis, definiu-se o operador permutação. Esse operador, como o nome diz, permuta operações na mesma máquina.

C1 | 3 9 1 6 | 5 2 7 | 11 13 4 | 12 10 8 |
* * (gene de permutação)

permutação

C1* | 3 6 1 9 | 5 2 7 | 11 13 4 | 12 10 8 |
* *

Assim sempre serão gerados indivíduos viáveis. É claro que se fosse permitido permutação entre máquinas teríamos indivíduos inviáveis. É bom lembrar que os pontos de permutação são aleatórios.

4.2.2. Cruzamento

O processo de cruzamento também só ocorre entre os sub-cromossomos das mesmas máquinas para evitar soluções inviáveis. O algoritmo de cruzamento utilizado será descrito por meio de um exemplo.

Passo 1 - Seleciona-se dois pais P1 e P2.

P1 | 9 6 1 3 | 5 2 7 | 11 13 4 | 12 10 8 |
P2 | 3 9 6 1 | 2 5 7 | 13 11 4 | 10 8 12 |

Passo 2 - Sorteia-se um sub-cromossomo e nesse sub-cromossomo sorteia-se o ponto de cruzamento em P2.

P1 | 9 6 1 3 | 5 2 7 | 11 13 4 | 12 10 8 |
P2 | 3 9 6 1 | 2 5 7 | 13 11 4 | 10 8 12 |
*

Passo 3 - Retira-se a tarefa que ocupa a posição sorteada em P2 de P1 deixando um buraco, que será representado pela letra "B".

P1 | 9 B 1 3 | 5 2 7 | 11 13 4 | 12 10 8 |
P2 | 3 9 6 1 | 2 5 7 | 13 11 4 | 10 8 12 |
*

Passo 4 - Desloca-se o buraco B de P1 até que ele fique no mesmo gene do elemento sorteado em P2

P1 | 9 1 B 3 | 5 2 7 | 11 13 4 | 12 10 8 |
P2 | 3 9 6 1 | 2 5 7 | 13 11 4 | 10 8 12 |
*

Passo 5 - Capta-se o material do gene sorteado do P2 e insere no B de P1, gerando o indivíduo filho C1.

P1 | 9 1 B 3 | 5 2 7 | 11 13 4 | 12 10 8 |
P2 | 3 9 6 1 | 2 5 7 | 13 11 4 | 10 8 12 |

⇩

C1 | 9 1 6 3 | 5 2 7 | 11 13 4 | 12 10 8 |

Passo 6 - O resultado é um filho C1

Passo 7 – Foi obtido o filho C1. Voltando para o passo 2 e trocando os índices de 2 para 1 e 1 para 2 é obtido o filho C2.

A seleção dos indivíduos do cruzamento foi feita pelo método da roleta que pode ser encontrado em [MZ96] e [GB89].

4.3. Função de desempenho

A função de desempenho é a própria função objetivo: minimizar o *makespan*. O código utilizado foi o fornecido em [ZIV04] com um pequena alteração como descrito na seção 3.2

4.4. Detecção de Ciclos no Indivíduo (DCI)

Como optou-se por trabalhar com operadores livres, fez-se necessário implementar um algoritmo para detecção de indivíduos inviáveis e a criação de novos.

Como a codificação real é usada, tanto a permutação quanto o cruzamento podem gerar indivíduos inviáveis em relação a ordem de precedência das atividades, daí a ncessidade do DCI.

O código utilizado para detecção de ciclos foi obtido em [ZIV04].

5. IMPLEMENTAÇÃO

A implementação foi realizada na linguagem C padrão sendo composta por dois blocos principais, um responsável por carregar os dados do problema a partir de um arquivo e o outro por encontrar a melhor solução para o problema através do algoritmo proposto.

5.1. O Algoritmo Genético

Baseando em [CTV95], [CV95] e [MZ96], optou-se por implementar o seguinte algoritmo:

(1) Gera-se uma população aleatória com N indivíduos.
(2) Percorre toda a população eliminando indivíduos inviáveis e criando novos com o DCI.
(3) Calcula-se o desempenho de cada indivíduo na população.
(4) Selecionam-se M melhores indivíduos para a próxima geração. (Elitismo).
(5) Com os indivíduos restantes implementa-se a roleta de acordo com a probabilidade de cruzamento[GB89].
(6) Cruzam-se os indivíduos selecionando-os e executa o DCI.
(7) Retiram-se P indivíduos com o maior *makespan*.
(8) Faz-se a mutação de P indivíduos aleatórios de acordo com a probabilidade de mutação.
(9) Os novos indivíduos somados aos selecionados no elitismo fazem parte da próxima geração.
(10) Se a iteração for maior que ITMAX (máximo número de iterações), termina a execução. Se não volte para (3)

O critério de parada utilizado foi o número de interações, definido no início do programa.

6. RESULTADOS COMPUTACIONAIS

Foram feitas execuções de algumas instâncias do problema e os resultados mais importantes são relacionados na tabela 3.

As instâncias mostradas nessa tabela constituem uma parte de um conjunto de instâncias que é freqüentemente usado para validar novas implementações para o problema de *Job Shop*.

Nessa tabela pode-se encontrar a solução ótima para a instância do problema, a melhor, a média e pior solução encontrada pela atual implementação para várias execuções do programa.

Os parâmetros utilizados no Algoritmo Genético foram população de 50 indivíduos, ITMAX igual a 1000 iterações, probabilidade de mutação de 5% e probabilidade de cruzamento de 60%.

Tabela 3

Prob.	*n x m*	Ótimo	Melhor	Média	Pior
ft06	6 x 6	55	55	57,2	66
la01	10 x 5	666	666	669,3	673
la02	10 x 5	655	655	659,6	662
la03	10 x 5	597	597	601,7	611
la04	10 x 5	590	590	594,2	606
la05	10 x 5	593	593	596,8	601
la06	15 x 5	926	926	930,3	944

O tempo médio de execução para as instâncias 6x6 foi de, aproximadamente, 2 minutos, para os 10x5, 3 minutos e para o 15x5, 15 minutos em um Athlon XP 1800+ com 256MB

7. CONCLUSÕES E TRABALHOS FUTUROS

Os resultados foram bem satisfatórios tanto em relação a obtenção do ótimo quanto ao tempo gasto para se obter as soluções. Na maior parte das execuções, a solução ótima foi encontrada. Para as execuções que o sequenciamento ótimo não foi obtido, uma solução bem próxima foi encontrada como poder ser comprovado pela média da tabela 3. Pode-se observar que o pior resultado também ficou próximo do valor ótimo.

Resultados melhores poderiam ser obtidos se fosse definido como critério de parada o tempo de

execução, mas optou-se pelo número de iterações com intuito de verificar a velocidade de convergência do algoritmo para a solução.

Comparando-se com as implementações em [NY91] e [CVT95], a implementação atual conseguiu resolver apenas instâncias menores. Isso acontece porque o Algoritmo Genético aqui proposto é puro, ou seja, não utiliza nenhum outro algoritmo para auxiliá-lo na busca. Enquanto em [CVT95], por exemplo, um método aproximado chamado *Priority Rules* é usado em conjunto com o Algoritmo Genético para melhorar as soluções.

Como sugestão para um próximo trabalho, a adição de conhecimento do problema ao Algoritmo Genético poderia levar a soluções melhores. Isto é, trabalhar com menos parâmentros probabilísticos .

Por exemplo, alterar o operador de permutação que faz trocas de tarefas aleatórias em máquinas direcionando-o para fazer trocas específicas das tarefas que estiverem no caminho crítico. A melhoria do sequenciamento aplicando essa técnica pode ser comprovada em [VAL95]. Além disso, a modificação do mecanismo de gerar a população inicial e uma implementação utilizando máquinas paralelas poder fornecer bons resultados.

É bom ressaltar que todas essas melhorias já estão em fase de implementação em busca de se obter resultados que possibilitem a submissão e, consequente, aceitação desse trabalho no Congresso da Sociedade Brasileira de Pesquisa Operacional (SOBRAPO 2004).

8. REFERÊNCIAS BIBLIOGRÁFICAS

[BZ96] Blazewicz, J., Ecker, K. H., Pesch, E., Shmidt, G. Weglarz, J. *Scheduling Computer and Manufacturing Process*, Springer, 1996.

[CTV95] Croce, F. D., Tadei, R., Volta, G., *A Genetic Algorithm for the Job Shop Problem*, Computers Operation Research, Vol. 22, No 1, pp 15-24, 1995.

[CV95] Chen, C., Vempati, V. S., Aljaber, N. *An application of Genetic Algorithms for Flow Shop Problems*, Journal of Operations Research, ed. 80, 389-396, 1995.

[GB89] Goldberg, D. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[GS78] Gonzalez, T. Sahni, S., *Flowshop and jobshop schedules:Complexity and aproximation*, Oper. Res. 20, 1978.

[LAL92] Van Laarhoven, P. J. M. Arts, E. H. L. Lenstra, J. K., *Job Shop scheduling by simulated annealing*, 113-125. 1992

[LL93] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B., *Sequencing and scheduling: algorithms and complexity*. Handbooks in Oper. Res. And Management Sci., Vol. 4, Logistics of Production and Inventory, Elsevier, Amsterdam, 1993.

[MZ96] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Third Edition, 1996.

[NY91] Nakano, R. e Yamada, T. *Conventional genetic algorithm for job shop scheduling*. Proceedings of the Fourth International Conference on Genetic Algorithms and Their Application, 1991

[RS64] Roy, B. e Sussman B., *Les problémes d'ordonnancement avec contraintes disjonctives*, SEMA, Note D.S. No. 9, Paris, 1964.

[TA93] Trubian, Dell'Amico, *Applying tabu search to the job shop scheduling problem*, Ann. Ops. Res. 41, 231-252. 1993.

[ZIV04] Ziviani, N., *Projeto e Análise de Algoritmos com implementações em Pascal e C*, 2a. edição, Editora Pioneira Thomson Learning, 2004.