ALGORITMOS GENÉTICOS PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Matheus Nohra Haddad[1], **Marcone Jamilson Freitas Souza**[1], **Haroldo Gambini Santos**[1]

[1] Departamento de Computação – Universidade Federal de Ouro Preto (UFOP)
Campus Universitário, Morro do Cruzeiro, CEP 35400-000, Ouro Preto (MG), Brasil

mathaddad@gmail.com, marcone.freitas@gmail.com, haroldo.santos@gmail.com

Resumo. *Este trabalho tem seu foco em problemas de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. Tem-se como objetivo, neste trabalho, a minimização do tempo máximo de conclusão do sequenciamento, o chamado makespan. Visando a sua resolução, são implementados dois algoritmos genéticos, um baseado na representação por vetores de listas, denominado AGVL, e outro baseado na representação por chaves randômicas, denominado AGCR. São utilizados métodos gulosos, bem como métodos parcialmente gulosos na geração de soluções iniciais. Buscas locais também são utilizadas a fim de se obterem melhores resultados. Foram realizados testes computacionais que mostraram a robustez do algoritmo AGCR, mas que também mostraram que o algoritmo AGVL necessita de calibrações.*

PALAVRAS-CHAVE*: Sequenciamento em máquinas paralelas, Algoritmos Genéticos, Chaves Randômicas, makespan*

Abstract. *This work has its focus on unrelated parallel machine scheduling problems of sequence dependent setup times. It has been the objective, in this work, to minimize the maximum completion time of the schedule, called makespan. Aiming to its resolution, two genetics algorithms are implemented, one based on vector-with-lists representation, called AGVL, and another based on random keys representation, called AGCR. Greedy methods are used, as well partially greedy methods to generate initial solutions. Local searches are also used in order to obtain better solutions. Computational tests were executed and they showed a robustness of the algorithm AGCR, but they also showed that the algorithm AGVL needs calibrations.*

KEYWORDS*: Parallel machine scheduling, Genetics Algorithms, Random Keys, makespan*

1 INTRODUÇÃO

Neste trabalho estuda-se o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência (*Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times*), ou apenas UPMSP. O objetivo abordado neste trabalho é o de minimizar o tempo total de conclusão do sequenciamento, o chamado *makespan*. Este problema tem importância tanto teórica quanto prática. Teórica, pois é um problema de difícil solução por pertencer à classe NP-difícil (Ravetti et al., 2007) e prática, pois existem muitas situações onde ele aparece, como, por exemplo, fabricações em indústrias têxteis (Pereira Lopes e de Carvalho, 2007).

Devido a dificuldade de encontrar soluções ótimas para o UPMSP em tempos aceitáveis, a utilização de heurísticas para se obter soluções de qualidade deve ser considerada. São encontradas na literatura várias abordagens heurísticas que procuram resolver problemas semelhantes e que serviram de inspiração para o desenvolvimento deste trabalho.

Alguns autores abordam problemas similares ao UPMSP. Logendran et al. (2007) utilizaram uma Busca Tabu com o objetivo de minimizar o tempo total de atrasos com pesos. Em Weng et al. (2001) são propostas sete heurísticas com o intuito de minimizar o tempo total de atrasos com pesos. Já em Kim et al. (2002) é utilizado o método *Simulated Annealing* objetivando a minimização do tempo total de atraso. Kim et al. (2003) tratam um problema parecido, com datas de entrega iguais, e implementam quatro heurísticas para tentar solucioná-lo.

Randall e Kurz (2007) propuseram um algoritmo genético adaptativo para o problema envolvendo datas de entregas e cujo objetivo é minimizar o tempo total de atraso com pesos. Os autores representam os indivíduos com chaves randômicas. A população inicial é gerada aleatoriamente e a seleção para reprodução é feita escolhendo dois indivíduos aleatoriamente da população. São utilizados quatro operadores de cruzamento: um ponto de corte, dois pontos de corte, uniforme e uniforme parametrizado. As aplicações destes operadores são adaptadas de acordo com a evolução, sendo que aqueles que geram melhores indivíduos são escolhidos. Em cada geração, 20% da população é gerada pela reprodução elitista, 79% pelo cruzamento e 1% por imigração.

Visando a minimização do *makespan* são encontradas poucas referências, dentre elas, de Paula et al. (2007) e Vallada e Ruiz (2011). No primeiro trabalho é implementado o método *Variable Neighbourhood Search* para resolver problemas em máquinas não-relacionadas (UPMSP) e também idênticas, sendo impostas datas de entrega para cada tarefa e penalidades pelo atraso nas datas de entrega. No segundo trabalho é tratado apenas o UPMSP, o qual é resolvido por meio de Algoritmos Genéticos. No algoritmos desses autores Vallada e Ruiz (2011), a população inicial é criada com um indivíduo pela heurística de múltipla inserção Kurz e Askin (2001) e o resto da população é gerado aleatoriamente. Depois de criados, são aplicadas buscas locais em todos os indivíduos. A seleção para o cruzamento é realizada por torneio n-ário. Nos cruzamentos são realizados procedimentos de buscas locais limitadas. Buscas locais baseadas em movimentos de múltipla inserção entre máquinas também são aplicadas a todos os indivíduos. A seleção para a sobrevivência é feita pela estratégia estacionária, incluindo indivíduos originais na população, desde que sejam melhores que os piores. Dois algoritmos, GA1 e GA2, que se diferem pelos parâmetros, foram testados. O algoritmo GA2 obteve os melhores resultados.

Inspirado em Vallada e Ruiz (2011) e Randall e Kurz (2007), são desenvolvidos

no presente trabalho dois algoritmos genéticos, nomeados AGVL e AGCR, na tentativa de encontrar soluções de melhor qualidade para o UPMSP. O primeiro algoritmo implementa a representação usual, utilizando vetores de listas, e o segundo implementa a representação baseada em chaves randômicas. Além de se obter algoritmos robustos, o objetivo deste trabalho é o de estudar o comportamento da representação por chaves randômicas, ainda não abordada na literatura para esse problema, de nosso conhecimento.

O restante deste trabalho está estruturado como segue. Na Seção 2 o problema em questão é caracterizado. A Seção 3 apresenta a metodologia utilizada para o desenvolvimento deste trabalho. Nas Seções 4 e 5 são apresentados os algoritmos genéticos propostos para resolver o UPMSP. Resultados computacionais são apresentados na Seção 6. Finalmente, na Seção 7, conclui-se este trabalho, bem como são apresentadas possíveis propostas a serem exploradas em trabalhos futuros.

2 CARACTERIZAÇÃO DO PROBLEMA

No problema de sequenciamento em máquinas paralelas não-relacionadas tem-se um conjunto $N = \{1, ..., n\}$ de n tarefas e um conjunto $M = \{1, ..., m\}$ de m máquinas não-relacionadas, com as seguintes características: (a) Cada tarefa deve ser processada exatamente uma vez por apenas uma máquina; (b) Cada tarefa j possui um tempo de processamento p_{ij} que depende da máquina i na qual será alocada. Por esta característica, as máquinas são ditas não-relacionadas; (c) Existem tempos de preparação entre as tarefas, s_{ijk}, em que i representa a máquina cujas tarefas j e k serão processadas, nesta ordem. Tais tempos de preparação são dependentes da sequência e da máquina. O objetivo é encontrar um sequenciamento das n tarefas nas m máquinas de forma a minimizar o tempo máximo de conclusão do sequenciamento, o chamado *makespan* ou $C_{\max}$. Pelas características citadas, o UPMSP é definido como $R|S_{ijk}|C_{\max}$ (Pinedo, 2008).

A Tabela 1 contém os tempos de processamento de sete tarefas em duas máquinas. Na Tabela 2 estão contidos os tempos de preparação para as máquinas M1 e M2. A Figura 1 ilustra um possível sequenciamento para esse exemplo.

Tabela 1. Tempos de processamento das máquinas M1 e M2

	1	2	3	4	5	6	7
M1	20	25	28	17	43	9	65
M2	4	21	15	32	38	23	52

Tabela 2. Tempos de preparação das máquinas M1 e M2

M1	1	2	3	4	5	6	7	M2	1	2	3	4	5	6	7
1	0	1	8	1	3	9	6	1	0	4	6	5	10	3	2
2	4	0	7	3	7	8	4	2	1	0	6	2	7	7	5
3	7	3	0	2	3	5	3	3	2	6	0	6	8	1	4
4	3	8	3	0	5	2	2	4	5	7	1	0	12	10	6
5	8	3	7	9	0	5	7	5	7	9	5	7	0	4	8
6	8	8	1	2	2	0	9	6	9	3	5	4	9	0	3
7	1	4	5	2	3	5	0	7	3	2	6	1	5	6	0

Na Figura 1 observa-se, por exemplo, que a tarefa 6 é alocada à máquina M2 na terceira posição, com tempo de processamento $p_{26} = 23$, tendo a tarefa 4 como predecessora e a tarefa 3 como sucessora. As partes hachuradas da figura representam os tempos de

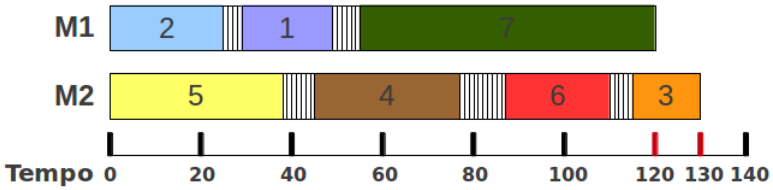

Figura 1. Exemplo de um possível sequenciamento

preparação entre as tarefas. Assim, neste exemplo, são computados os tempos $s_{246} = 10$ e $s_{263} = 5$. Os tempos marcados em vermelho representam os tempos de conclusão de cada máquina. O tempo de conclusão da máquina M1 é 120 e o da máquina M2 é 130, o que resulta em *makespan* de 130 unidades de tempo.

3 METODOLOGIA

3.1 Representação da Solução

Serão implementadas duas representações de soluções para o UPMSP. A primeira usa um vetor v de m máquinas, com cada máquina representada por uma lista de tarefas. Em cada posição de v está representada a máquina e para cada máquina está associada uma lista, representando a sequência de processamento da máquina.

Na segunda implementação, representa-se uma solução por meio de chaves randômicas (Bean, 1994). Nessa representação se tem um vetor v, no qual cada posição j contém um número real aleatório advindo da distribuição $U[1.00, m + 1)$, sendo m o número de máquinas. A posição j representa a tarefa e o número real indica em qual máquina a tarefa será executada, indicada pela parte inteira, e em qual ordem, indicada pela parte fracionária. Deve-se ordenar os números reais para saber a ordem de sequenciamento. Desta forma, o vetor $v = [1.45, 1.32, 2.95, 2.54, 2.36, 2.71, 1.79]$ pode ser uma representação do sequenciamento ilustrado pela Figura 1.

A utilização de chaves randômicas facilita as operações de cruzamento e mutação, gerando somente soluções viáveis, diferentemente da primeira representação, onde se devem realizar procedimentos para retirar tarefas duplicadas.

3.2 Avaliação de Uma Solução

Uma solução v tem como valor de avaliação o tempo de processamento da máquina que irá concluir suas tarefas por último, ou seja, o *makespan*.

3.3 Algoritmos Genéticos

Para resolver o UPMSP, propõem-se dois algoritmos genéticos, baseados nas duas representações apresentadas. O primeiro, nomeado AGVL, utiliza a representação com vetor de listas e o segundo, nomeado AGCR, representa soluções na forma de chaves randômicas. Os dois algoritmos utilizam os mesmos parâmetros, distinguindo-se apenas na forma de representação. O Algoritmo 1 mostra o pseudocódigo deles.

Algoritmo 1: AGVL - AGCR

```
   Entrada: nInd, p_c, p_m, p_t, criterioParada
1  início
2      Pop ← {nInd};
3      para cada ind ∈ Pop faça
4          ind ← geraInd();
5          ind ← avalia();
6          ind ← buscaLocal();
7          atualizaMelhor();
8      fim
9      repita
10         pai1, pai2 ← selecionaParaRep(p_t, Pop);
11         r ← número real no intervalo [0, 1];
12         se r <= p_c então
13             filho1, filho2 ← crossover(pai1, pai2);
14         senão
15             filho1, filho2 ← pai1, pai2;
16         fim
17         r ← número real no intervalo [0, 1];
18         se r <= p_m então
19             filho1, filho2 ← mutacao();
20         fim
21         filho1, filho2 ← avalia();
22         filho1, filho2 ← buscaLocal();
23         atualizaMelhor();
24         P_d, P_e ← selecionaParaSob(Pop);
25         P_d ← filho1;
26         P_e ← filho2;
27     até que o critério de parada seja satisfeito ;
28 fim
```

O Algoritmo 1 recebe como parâmetros: 1) *nInd*, que representa o números de indivíduos da população; 2) p_c, um número real entre 0 e 1, que representa a probabilidade de cruzamento; 3) p_c, um número real entre 0 e 1 representando a probabilidade de mutação; 4) p_t, um número real entre 0 e 1 que representa a porcentagem da população que participará do torneio; 5) critério de parada, que nos algoritmos implementados foi o tempo de processamento.

Na linha 4 do Algoritmo 1 são gerados os indivíduos. Logo após, na linha 5, eles são avaliados e passam por um processo de busca local (linha 6), sempre atualizando o melhor indivíduo. De posse da população inicial é iniciado o processo evolutivo, que é executado até um certo critério de parada estabelecido. Na linha 10 são selecionados dois indivíduos para a reprodução. Tais indivíduos são submetidos a um cruzamento com uma probabilidade p_c (linhas 11 a 16). Após este processo, os indivíduos passam por um processo de mutação com uma probabilidade p_m, nas linhas 17 a 20. Nas linhas 21 e 22 os indivíduos resultantes são avaliados e sofrem uma busca local, atualizando o melhor indivíduo encontrado. Finalmente, é realizada uma seleção para saber quais indivíduos da população os novos indivíduos substituirão (linhas 24 a 26). Todas estas etapas são detalhadas a seguir.

4 ALGORITMO AGVL

4.1 Geração da População Inicial

A população inicial é gerada pela seguinte proporção: um indivíduo gerado pela heurística *Adaptive Shortest Processing Time* (ASPT), $(nInd/2 - 1)$ gerados por construção parcialmente gulosa e $nInd/2$ gerados aleatoriamente. Os métodos construtivos são apresentados a seguir.

Na construção pela heurística ASPT, primeiramente as tarefas são ordenadas em ordem crescente de tempos de processamento, ou seja, de acordo com a máquina na qual a tarefa tem seu menor tempo de processamento. A tarefa de menor tempo de processamento é alocada na respectiva máquina. Para as outras tarefas agora, deve-se ordenar também, em ordem crescente de tempos de processamento, porém nas máquinas em que já existirem tarefas alocadas, deve-se somar a esse tempo, os tempos de preparação. Aloca-se a tarefa que tiver o menor tempo total. Esse processo continua até que todas as tarefas sejam alocadas.

A construção parcialmente gulosa é feita de acordo com a fase de construção da heurística GRASP (Feo e Resende, 1995), usando como guia a função de avaliação g da heurística ASPT. A cada iteração é construída uma Lista Restrita de Candidatos (LRC) das tarefas melhor classificadas da Lista de Tarefas Candidatas (LC), isto é, da lista das tarefas ainda não alocadas. As tarefas j de LC são classificadas de acordo com a função guia g. Integram a LRC as tarefas j tais que $g(j) \leq g_{\min} + \alpha \times (g_{\max} - g_{\min})$, sendo $g_{\min}$ o menor tempo de término, $g_{\max}$ o maior tempo de término e α um parâmetro GRASP responsável por controlar o quão gulosa será a solução. A seguir, uma tarefa j é escolhida aleatoriamente de LRC e alocada à máquina respectiva. O processo é repetido até que todas as tarefas sejam alocadas.

4.2 Seleção Para Cruzamento

A seleção dos indivíduos é realizada por torneio n-ário. Nesse método de seleção, determina-se aleatoriamente uma parcela da população de tamanho p_t. A seguir, dentre essa parcela, obtém-se o indivíduo com o menor *makespan*. Esse processo é repetido para se obter o outro indivíduo. Esses dois indivíduos são utilizados no cruzamento.

4.3 Cruzamento

De posse dos dois indivíduos, advindos do método de seleção, o cruzamento é realizado com probabilidade p_c. Na Figura 2 o cruzamento é representado.

Utilizando o primeiro pai apenas, para cada máquina, um ponto de corte é escolhido aleatoriamente. As tarefas que estão antes do ponto de corte são copiadas no primeiro filho, na mesma ordem. As tarefas que estão depois do ponto de corte, são copiadas no segundo filho, na mesma ordem. Agora descarta-se o primeiro pai e utiliza-se somente o segundo pai. Primeiramente, as redundâncias são eliminadas, ou seja, as tarefas dos filhos repetidas no segundo pai são apagadas. Para cada tarefa restante, e na ordem em que estão, deve-se colocá-las em todas as posições da máquina respectiva e alocá-las onde se obtém o menor tempo de conclusão da máquina, procedimento conhecido como *Busca Local Limitada*, de acordo com Vallada e Ruiz (2011). Esse processo tem como resultado dois novos indivíduos.

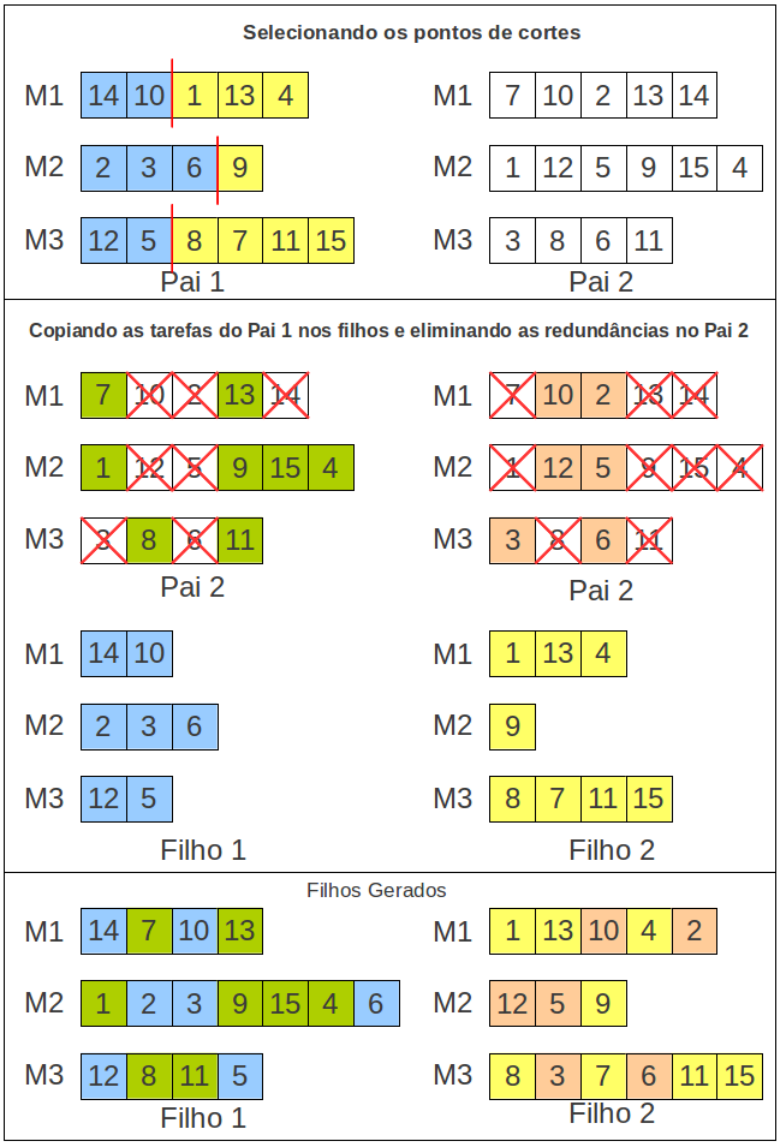

Figura 2. Cruzamento no AGVL (Adaptado de Vallada e Ruiz (2011))

4.4 Mutação

Com uma probabilidade p_m, a mutação é realizada com base em movimento de inserção. Seleciona-se, aleatoriamente, uma máquina e uma tarefa alocada nesta máquina. A seguir, uma nova posição em qualquer máquina é escolhida, também de forma aleatória, para alocar essa tarefa.

4.5 Busca Local

Para a Busca Local são utilizados movimentos de múltipla inserção com a estratégia *First Improvement*. Nesta busca, cada tarefa de cada máquina é inserida em todas as posições de todas as máquinas. O movimento é aceito se os tempos de conclusão das máquinas envolvidas são reduzidos. Caso o tempo de conclusão de uma máquina é reduzido e o tempo de conclusão da outra máquina é acrescido, o movimento também é aceito. Porém, neste caso, somente há aceitação se o valor de tempo reduzido for maior que o valor de tempo aumentado. Destaca-se que, mesmo na ausência de melhoria no valor do

makespan, o movimento pode ser aceito. Ao ocorrer a aceitação de um movimento, a busca é reiniciada e só termina quando se encontra um ótimo local, ou seja, quando não existir nenhum movimento de aceitação para a vizinhança de múltipla inserção.

4.6 Seleção Para Sobrevivência

A cada iteração evolutiva cada um dos dois indivíduos gerados substitui o pior indivíduo da população, desde que ele seja diferente dos demais indivíduos da população e tenha *makespan* menor que o indivíduo que ele substitui. Caso essas condições não sejam verificadas, o indivíduo é descartado.

5 ALGORITMO AGCR

5.1 Geração da População Inicial

A inicialização da população é feita pelos mesmos métodos utilizados no AGVL e segue a mesma proporção: um indivíduo gerado pela heurística ASPT, $(nInd/2 - 1)$ gerados pela construção parcialmente gulosa e $nInd/2$ gerados aleatoriamente. Contudo, nesse caso, deve ser realizado um procedimento de codificação dos indivíduos gerados pela ASPT e pela construção gulosa. Esse procedimento de codificação é explicado a seguir.

5.1.1 Codificação dos Indivíduos

Como a ideia da representação por chaves randômicas é justamente a aleatoriedade, propõe-se um algoritmo que visa o seu uso para codificar indivíduos gerados por heurísticas gulosas ou parcialmente gulosas. Números reais com seis casas decimais são utilizados.

Para cada máquina i e para cada tarefa j é gerado um número real r dentro da distribuição $U[i, i + 1)$, onde j é a primeira tarefa alocada na máquina i. Para a próxima tarefa k alocada em i, gera-se um número real r' dentro da distribuição $U[r, i + 1)$ e assim sucessivamente.

5.2 Seleção Para Cruzamento

A seleção para o cruzamento é realizada de forma aleatória, ou seja, aleatoriamente dois indivíduos da população são escolhidos. Se estes dois indivíduos forem iguais, um outro indivíduo é novamente escolhido até que sejam diferentes.

5.3 Cruzamento

Para realizar o cruzamento foi utilizado o operador de cruzamento uniforme parametrizado (*Parametric Uniform Crossover - PUC*). Durante esse cruzamento, para cada gene sorteia-se um número na distribuição $U(0, 1)$. Então, para cada gene é verificado se esse número sorteado é menor que a probabilidade de cruzamento p_c. Se esse número for menor que p_c, o primeiro filho herda o gene do primeiro pai e o segundo filho herda o gene do segundo pai, senão o primeiro filho herda o gene do segundo pai e o segundo filho herda o gene do primeiro pai. O PUC é ilustrado pela Figura 3, sendo $p_c = 0.8$.

5.4 Mutação

A mutação é realizada por um movimento de deslocamento. Aleatoriamente uma posição é escolhida para ser alterada, e, aleatoriamente, uma outra posição também é escolhida. Insere-se, então, o número real presente da primeira posição escolhida na segunda

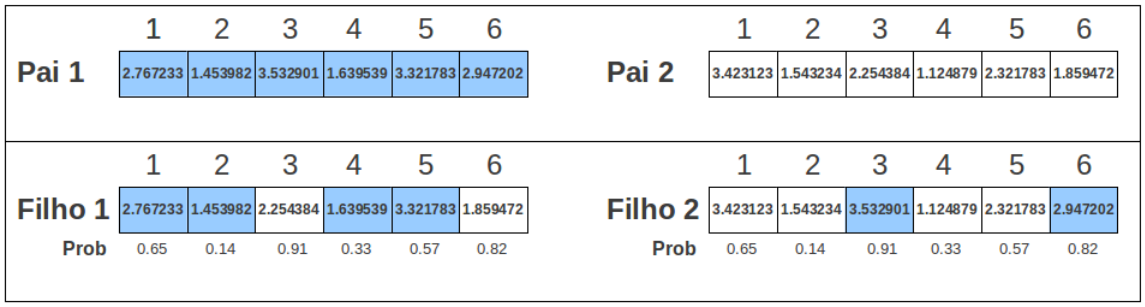

Figura 3. ***Parametric Uniform Crossover***

posição escolhida. Os números à direita da primeira posição até a segunda posição são deslocados à esquerda. A Figura 4 mostra este procedimento de mutação. Após este movimento, a tarefa 2 troca de posição com a tarefa 3 e esta, por sua vez, troca de posição com a tarefa 4, a qual vai para a posição da tarefa 2.

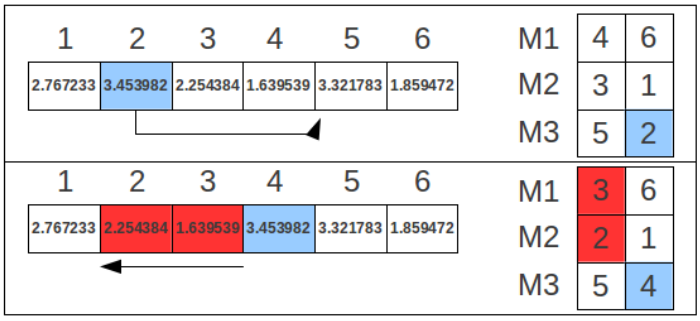

Figura 4. Mutação no AGCR

5.5 Busca Local

A busca local no AGCR é realizada utilizando-se os movimentos de troca e substituição com a estratégia *Best Improvement*. No movimento de troca os números reais são trocados, representando a troca de duas tarefas. A substituição é realizada com o intuito de alocar as tarefas em outras máquinas e isso é feito substituindo-se o número inteiro de uma máquina por outro representativo de outra máquina.

5.6 Seleção Para Sobrevivência

Cada indivíduo gerado é incluído na população, no lugar do pior indivíduo, apenas se não existirem indivíduos iguais a ele na população e se ele for melhor do que este pior.

6 RESULTADOS PARCIAIS

Para a realização dos testes computacionais, foi utilizado um conjunto de 640 problemas-teste da literatura de SOA (2011), envolvendo combinações de 6, 8, 10 e 12 tarefas e 2, 3, 4 e 5 máquinas. Em SOA (2011) também são fornecidos os melhores valores de *makespan* desses problemas-teste.

Os algoritmos AGVL e AGCR foram desenvolvidos na linguagem C++. Todos os experimentos foram executados em um computador com processador *Intel Core 2 Quad 2,4 GHz* com 4 GB de memória RAM e sistema operacional *Ubuntu 10.10*. Os parâmetros

utilizados no AGVL foram: ***nInd*** $= 200$, $p_c = 80\%$, $p_m = 5\%$ e $p_t = 10\%$. No AGCR os parâmetros ***nInd*** $= 200$, $p_c = 80\%$ e $p_m = 5\%$ foram utilizados. Na construção parcialmente gulosa, em ambos algoritmos, foi utilizado $\alpha = 0,2$. O critério de parada para ambos é o tempo máximo de duração do processamento $Tempo_{\max}$, em milissegundos, obtido pela Eq. (1). Para o parâmetro t foram testados três valores: 10, 30 e 50.

$$Tempo_{\max} = n \times (m/2) \times t \;\; ms \tag{1}$$

Na métrica utilizada para a comparação dos algoritmos, dada pela Eq. (2), o objetivo é verificar a variabilidade das soluções finais produzidas pelos algoritmos. Nessa medida, calcula-se, para cada algoritmo *Alg* aplicado a um problema-teste i, o desvio percentual relativo RPD_i da solução encontrada $\bar{f}_i^{Alg}$ em relação à melhor solução f_i^* conhecida até então. Os algoritmos AGVL e AGCR foram executados trinta vezes e, ao final das trinta execuções, calcula-se a média RPD_i^{avg} dos valores de RPD_i encontrados.

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \tag{2}$$

A Tabela 3 mostra, para todas as instâncias, o RPD_i^{avg} dos algoritmos AGVL, AGCR, bem como do algoritmo GA2 de Vallada e Ruiz (2011). Destaca-se que o GA2 foi executado em uma máquina similar, também com 2,4 GHz, possibilitando a comparação.

Tabela 3. Médias dos Desvios Percentuais Relativos dos algoritmos GA2, AGVL e o AGCR, com t = 10/30/50

Instâncias	GA2	AGVL	AGCR
6 x 2	0,00/0,00/0,00	0,04/0,00/0,03	0,00/0,00/0,00
6 x 3	0,07/0,08/0,08	0,45/0,49/0,34	0,00/0,00/0,00
6 x 4	0,16/0,37/0,27	0,90/0,64/0,74	0,00/0,00/0,00
6 x 5	0,10/0,12/0,21	0,67/0,82/0,78	0,00/0,00/0,00
8 x 2	0,03/0,00/0,03	0,18/0,19/0,17	0,00/0,00/0,00
8 x 3	0,32/0,31/0,24	0,54/0,58/0,59	0,00/0,00/0,00
8 x 4	0,50/0,41/0,39	0,92/0,90/0,94	0,00/0,00/0,00
8 x 5	0,23/0,11/0,20	0,32/0,29/0,31	0,00/0,00/0,00
10 x 2	0,19/0,07/0,17	0,05/0,06/0,10	0,02/0,01/0,01
10 x 3	0,15/0,18/0,20	0,23/0,12/0,07	0,03/0,01/0,03
10 x 4	0,26/0,30/0,32	0,22/0,37/0,39	0,09/0,08/0,09
10 x 5	1,15/1,03/1,15	0,83/0,62/0,52	0,08/0,06/0,07
12 x 2	0,15/0,10/0,09	0,38/0,31/0,32	0,57/0,57/0,53
12 x 3	0,15/0,12/0,08	0,09/0,04/0,09	0,45/0,43/0,47
12 x 4	1,00/0,89/0,75	0,68/0,49/0,46	0,96/0,63/0,48
12 x 5	1,54/1,49/1,50	1,04/1,13/0,69	1,40/1,24/0,77
RPD^{avg}	0,37/0,35/0,36	0,47/0,44/0,41	0,23/0,19/0,15

O algoritmo AGVL não obteve boas médias, provavelmente por conta das buscas locais executadas ao longo da fase de evolução do algoritmo genético. Já o algoritmo AGCR encontra sempre o melhor valor para todas as instâncias com 6 e 8 tarefas. Nas instâncias com 10 e 12 tarefas as médias encontradas são superiores às encontradas pelo GA2, exceto nas instâncias com 12 tarefas e 2 e 3 máquinas. Em geral, o AGCR obteve médias melhores que os outros algoritmos.

Para poder comparar os algoritmos AGVL e AGCR implementados, foi realizado um teste de probabilidade empírica (Aiex et al., 2002). Para execução dos experimentos, usou-se um problema-teste com 12 tarefas e 5 máquinas, tendo como alvo a melhor solução conhecida. Os dois algoritmos foram aplicados 120 vezes e sempre que o alvo era alcançado, eles eram interrompidos e o tempo era registrado. Esses tempos foram, então, ordenados de forma crescente e, para cada tempo t_i foi associada uma probabilidade $p_i = (i - 0,5)/120$. Os resultados do gráfico $t_i \times p_i$ são apresentados na Figura 5.

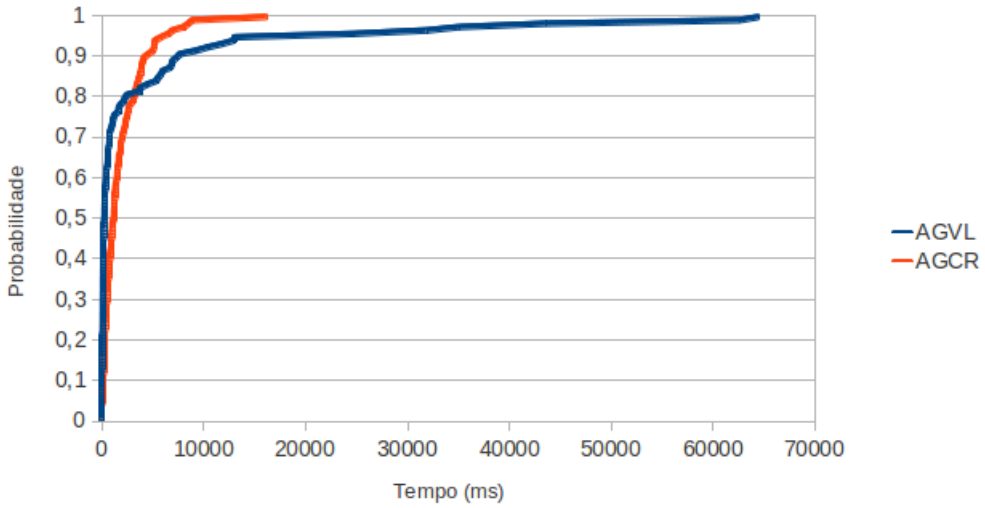

Figura 5. Teste de Probabilidade Empírica

Nos instantes iniciais, o AGVL tem melhor desempenho. No entanto, por volta de 5 segundos, o AGCR ultrapassa o AGVL, em termos de desempenho. Além disso, ele alcança o valor alvo, com quase 100% de probabilidade, em cerca de 10 segundos; enquanto o AGVL leva mais de 1 minuto para alcançar esse alvo.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve seu foco no problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. O objetivo almejado é a minimização do *makespan*.

Foram propostos dois algoritmos genéticos para a resolução desse problema, um baseado na representação de uma solução por vetores de listas (AGVL) e o outro baseado na representação de uma solução por chaves randômicas(AGCR). São utilizadas estratégias gulosas e parcialmente gulosas na geração de soluções iniciais, bem como buscas locais para melhorar a qualidade da população inicial.

Analisando os resultados obtidos e as métricas para as comparações adotadas, observa-se que o algoritmo AGCR obteve uma maior robustez em relação aos demais, pois obteve menor variabilidade das soluções encontradas. Já o algoritmo AGVL não obteve bons resultados, provavelmente por contas das buscas locais exaustivas.

Como trabalhos futuros, propõe-se testar os algoritmos nos problemas-teste maiores disponíveis em SOA (2011), assim como calibrar melhor seus parâmetros, além de implementar uma estratégia de Reconexão por Caminhos em ambos os algoritmos, visando a encontrar melhora nas soluções.

Agradecimentos

Os autores agradecem à UFOP, ao CNPq e à FAPEMIG pelo apoio ao desenvolvimento deste trabalho.

Referências

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, v. 8, p. 343–373.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, v. 6, p. 154–160.

de Paula, M. R.; Ravetti, M. G.; Mateus, G. R. e Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. *IMA Journal of Management Mathematics*, v. 18, p. 101–115.

Feo, T. e Resende, M. (1995). Greedy randomized search procedure. *Journal of Global Optimization*, v. 6, p. 109–133.

Kim, D. W.; Kim, K. H.; Jang, W. e Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, v. 18, p. 223–231.

Kim, D. W.; Na, D. G. e Frank Chen, F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, v. 19, p. 173–181.

Kurz, M. e Askin, R. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *Int. Journal of Production Research*, v. 39, p. 3747–3769.

Logendran, R.; McDonell, B. e Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations research*, v. 34, n. 11, p. 3420–3438.

Pereira Lopes, M. J. e deCarvalho, J. M. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European journal of operational research*, v. 176, p. 1508–1527.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems*. Springer Verlag.

Randall, P. A. e Kurz, M. E. (2007). Effectiveness of Adaptive Crossover Procedures for a Genetic Algorithm to Schedule Unrelated Parallel Machines with Setups. *Int. Journal of Operations Research*, v. 4, p. 1–10.

Ravetti, M. G.; Mateus, G. R.; Rocha, P. L. e Pardalos, P. M. (2007). A scheduling problem with unrelated parallel machines and sequence dependent setups. *International Journal of Operational Research*, v. 2, p. 380–399.

SOA,. Instâncias para o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência, Acesso em 21 de abril(2011). URL `http://soa.iti.es/problem-instances`.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*. To appear. doi:10.1016/j.ejor.2011.01.011.

Weng, M. X.; Lu, J. e Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, v. 70, p. 215–226.