# ALGORITMO GENÉTICO APLICADO À MINIMIZAÇÃO DE MAKESPAN EM MÁQUINAS PARALELAS NÃO RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

**Rodney O. M. Diana**[1], **Eduardo C. de Siqueira**[1], **Moacir F. F. Filho**[1], **Sérgio R. de Souza**[1]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Av. Amazonas, 7675, CEP 30510-000, Belo Horizonte (MG), Brasil

rodneyoliveira@dppg.cefetmg.br, eduardosiqueira@dppg.cefetmg.br

franca@des.cefetmg.br, sergio@dppg.cefetmg.br

***Resumo.*** *Este trabalho apresenta três variações de um algoritmo genético para resolução do problema de sequenciamento de tarefas em máquinas paralelas não relacionadas, com o objetivo de minimização do* makespan*. Tempos de preparação são considerados dependentes da máquina e da sequência de processamento das tarefas. A população inicial é construída por meio de uma abordagem GRASP. Para o processo de seleção, é utilizado o operador n-tournament, enquanto a operação de crossover é realizada por uma variação do operador One Point Order Crossover para sequenciamento em máquinas paralelas, denominado LSEC (Local Search Enhanced Crossover). Na mutação, é utilizado o operador de deslocamento de tarefas, amplamente utilizado para problemas de sequenciamento. Após a mutação, é aplicado um processo de busca local, sendo propostos dois métodos, que consideram características próprias da minimização do* makespan*. Foram realizados experimentos com um conjunto de instâncias da literatura, e os resultados alcançados são superiores aos conhecidos para todas as instâncias testadas.*
**PALAVRAS-CHAVE:** *Sequenciamento, Máquinas Paralelas,* Makespan*, Algoritmo Genético, GRASP, VND*

***Abstract.*** *This work shows three variations of a genetic algorithm to solve the unrelated parallel machines scheduling problem in order to minimize the makespan. Setup times between jobs are sequence and machine dependent. The construction of the initial population is made from GRASP algorithm. For the selection operator the n-tournament method is used, whereas the LSEC (Local Search Enhanced Crossover) method is used as the crossover operator. In the mutation step, the shift mutation, i.e., the most usual operator for scheduling problems, is used . After the mutation step, the offspring individuals are subject to a local search and two methods considering makespan minimization characteristics are proposed. Experiments conducted over a benchmark set of instances show that the algorithm proposed in this research outperforms the methods known in the literature, for all considered instances.*
**KEYWORDS:** *Scheduling, Parallel machines, Makespan, Genetic Algorithm, GRASP, VND*

## 1 Introdução

O problema de planejamento da produção tratado no presente trabalho consiste em definir: (i) a atribuição de um conjunto $N = \{1, ..., n\}$ de $n$ tarefas independentes a um conjunto $M = \{1, ..., m\}$ de $m$ máquinas paralelas não-relacionadas e continuamente disponíveis; (ii) a sequência de processamento das tarefas em cada máquina; e (iii) o instante de conclusão do processamento de cada tarefa, com o objetivo de minimizar o máximo instante de conclusão de todas as tarefas (*makespan*). Cada tarefa $j$ envolve uma única operação a ser realizada em uma única máquina $i$, demandando um processamento que não pode ser interrompido, uma vez tenha sido iniciado, e cuja duração $p_{ij}$ depende da máquina $i$ escolhida para seu processamento (situação que caracteriza as máquinas como não-relacionadas). A transição entre duas tarefas $j$ e $k$, adjacentes na sequência de tarefas atribuídas à máquina $i$, impõe um tempo de preparação $s_{ijk}$, que depende da sequência de processamento das tarefas, bem como da máquina $i$ em que as duas tarefas são processadas.

Tempos de preparação dependentes da máquina e da sequência de processamento das tarefas estão presentes em uma grande variedade de ambientes de produção. Entretanto, apenas recentemente esta condição passou a despertar, de modo mais generalizado, o interesse dos pesquisadores (Allahverdi et al., 2008). Em sua maioria, os artigos publicados até 1999 limitaram-se a resolver problemas nos quais os tempos de preparação eram considerados independentes da sequência ou eram simplesmente desconsiderados (Allahverdi e Gupta, 1999).

O instante de conclusão da última tarefa atribuída à máquina $i$ é representado por $C_i$. A minimização do máximo $C_i$, representado por $C_{max}$ e conhecido na literatura como *makespan*, é o objetivo mais estudado nos diversos problemas de planejamento da produção (Vallada e Ruiz, 2011).

Utilizando a notação apresentada em Pinedo (2008), o problema tratado no presente trabalho pode ser representado por $R|S_{ijk}|C_{max}$. Para resolvê-lo, sugere-se o aprimoramento do algoritmo genético proposto por Vallada e Ruiz (2011). Como uma das contribuições do presente trabalho, propõe-se a utilização da fase construtiva da meta-heurística GRASP (Feo e Resende, 1995) para a criação das soluções da população inicial. Outra contribuição consiste na utilização de um operador de busca local em vizinhanças variáveis (VND) (Mladenovic e Hansen, 1997), que leva em consideração as características próprias da minimização do *makespan* em máquinas paralelas. Com isso, consegue-se acelerar a convergência do algoritmo.

O restante deste artigo está organizado da seguinte forma. Na Seção 2 é apresentada uma revisão bibliográfica sobre o problema de sequenciamento de tarefas em máquinas paralelas, com tempos de preparação dependentes da sequência. A Seção 3 detalha o algoritmo genético e as adaptações propostas. Os experimentos realizados, o conjunto de instâncias e os resultados computacionais são apresentados na Seção 4. Por fim, na Seção 5 são apresentadas as conclusões sobre o trabalho realizado.

## 2 Revisão Bibliográfica

Um dos trabalhos pioneiros sobre sequenciamento de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência (PSMPSD) é Parker et al. (1977). No trabalho em questão, os autores consideram que as máquinas são idênticas e o objetivo é a minimização da soma dos custos de preparação. Um modelo matemático baseado no

problema de roteamento de veículos é apresentado e o algoritmo heurístico proposto por Clarke e Wright (1964) para roteamento de veículos é empregado.

Uma heurística construtiva, seguida de uma busca local com movimentos de troca, é apresentada em Guinet (1991) para o PSMPSD, em um ambiente de máquinas não relacionadas. São considerados dois objetivos: (i) minimização da média dos instantes de conclusão; e (ii) minimização do atraso médio. É proposta uma heurística, baseada em diversos trabalhos anteriores sobre roteamento de veículos, com o objetivo de minimizar o *makespan* e a média dos tempos de conclusão das tarefas, em um ambiente de máquinas idênticas.

Um algoritmo de busca tabu é proposto em França et al. (1996) para a minimização do *makespan* em máquinas idênticas. Em Gendreau et al. (2001), é proposto um limitante inferior para o problema tratado em França et al. (1996) e apresentada uma heurística, denominada *Divide and Merge*, que se mostra mais rápida que a implementação em busca tabu proposta em França et al. (1996) e com eficiência similar. Ainda para este mesmo problema, Kurz e Askin (2001) apresentam um modelo de programação matemática, as heurísticas *Multiple MULTI-FIT*, *Multiple Insertion* (MI) e *Slicing*, além de um algoritmo genético denominado GAK.

Alguns trabalhos com algoritmos exatos foram propostos para o sequenciamento em máquinas paralelas e não relacionadas. Contudo, soluções ótimas são encontradas apenas para instâncias com pequeno número de máquinas e tarefas. Um algoritmo *Branch and Bound* (B&B), operando em conjunto com uma relaxação lagrangeana, é apresentado em Martello et al. (1997) para a determinação de limitantes inferiores para problemas em que o objetivo é a minimização do *makespan*. Liaw et al. (2003) propõem um B&B para minimizar a soma ponderada dos atrasos. É proposta uma função para um limitante inferior e uma heurística para um limitante superior. Em Lancia (2000), é apresentada uma abordagem B&B para minimização do *makespan* em duas máquinas paralelas não relacionadas e que podem não estar disponíveis no instante inicial do horizonte de planejamento. Em nenhum destes trabalhos são considerados os tempos de preparação entre tarefas. Em Rocha et al. (2008), os tempos de preparação são considerados em um algoritmo B&B para a minimização da soma ponderada dos atrasos. A heurística GRASP é utilizada para determinar um limitante superior. O algoritmo mostra-se mais eficiente que dois modelos de programação matemática adaptados dos modelos propostos por Manne (1960) e Wagner (1959).

Conforme pode ser observado em Allahverdi et al. (2008), na última década houve uma intensa utilização de metaheurísticas para a solução do problema de sequenciamento em máquinas paralelas não relacionadas. Em Kim et al. (2002), é apresentada uma implementação de *Simulated Annealing* para tratar um problema de sequenciamento de lotes de tarefas, com o objetivo de minimizar a soma dos atrasos. A minimização da soma ponderada dos atrasos de lotes de tarefas é considerada em Kim et al. (2003). Em Al-Salem (2004), é apresentada uma heurística denominada *Partition Heuristic*, para o $R|S_{ijk}|C_{max}$. Em Helal e Al-Salem (2006), Rabadi et al. (2006) e de Paula et al. (2007) são apresentadas para o mesmo problema as meta-heurísticas Busca Tabu, Meta-RaPS e VNS, respectivamente. Ainda para este problema, Arnaout et al. (2010) apresenta uma abordagem através da meta-heurística Colônia de Formigas, enquanto Ying et al. (2012) apresenta uma abordagem via *Simulated Annealing*. Vallada e Ruiz (2011) apresenta um algoritmo genético e compara os resultados com as heurísticas MI e GAK de Kurz e Askin (2001) e Meta-RaPS

de Rabadi et al. (2006), mostrando que o algoritmo por eles proposto apresenta desempenho consideravelmente superior. A busca de Vallada e Ruiz (2011) é refinada em Coelho et al. (2012), o qual apresenta uma abordagem via processamento paralelo para o método, passando assim a apresentar desempenho amplamente superior.

## 3 Algoritmo Proposto

### 3.1 Representação de um indivíduo e construção da população inicial

Um indivíduo (solução para o problema de sequenciamento) é representado como um vetor de $m$ posições, cada uma delas associada a uma máquina. Por sua vez, cada posição do vetor de máquinas contém um apontador para um vetor de tarefas, em que a posição da tarefa representa a ordem na qual ela é processada.

A população inicial, formada por $P$ indivíduos, é representada por $P_0$. No presente trabalho, a fase construtiva do GRASP é utilizada na geração da população inicial. A construção de uma solução consiste na alocação das $n$ tarefas, uma a uma, em uma das $m$ máquinas, definindo soluções parciais $\bar{s}$. A escolha da próxima tarefa a ser incorporada à solução parcial $\bar{s}$ é feita aleatoriamente, com probabilidade uniforme, a partir de uma lista restrita de candidatos (LRC), composta por um subconjunto de tarefas ainda não alocadas e que apresentam as melhores avaliações, segundo a estratégia gulosa descrita a seguir. Realizada a escolha aleatória de uma tarefa em LRC, esta é inserida em $\bar{s}$, na posição definida pela estratégia gulosa. Em seguida, o conjunto $\bar{N}$ de tarefas ainda não alocadas é atualizado, retirando-se a tarefa recém-alocada na solução parcial. Este processo é repetido, até que o conjunto $\bar{N}$ fique vazio.

A estratégia gulosa é definida como: para cada tarefa $j \in \bar{N}$, avaliam-se os acréscimos no *makespan* decorrentes de sua possível inserção em cada uma das posições da solução parcial $\bar{s}$. Seja $\bar{s}_j$ a solução parcial $\bar{s}$ acrescida da tarefa $j$, na posição que impõe o menor acréscimo no *makespan*. As tarefas $j \in \bar{N}$ são ordenadas segundo valores não decrescentes do *makespan* de $\bar{s}_j$, ficando LRC composta pelas $q$ primeiras tarefas deste conjunto ordenado.

### 3.2 Seleção, *Crossover* e Mutação

O método de seleção através de torneios é muito utilizado em algoritmos genéticos clássicos. No presente trabalho, uma variação deste método, denominada *n-tournament*, é utilizada. Para cada par de indivíduos selecionados, são realizados $n$ torneios, definidos por um parâmetro denominado *"pressure"*. O par de indivíduos que for o vencedor destes torneios é selecionado para o processo de *crossover*. Em cada torneio, um indivíduo é escolhido aleatoriamente na população e disputa com o indivíduo vencedor do torneio anterior. No primeiro torneio, os dois indivíduos que o disputam são selecionados aleatoriamente. Para definir o individuo vencedor do torneio, é utilizado o valor do *makespan*, sendo que empates são desfeitos de maneira aleatória. São selecionados no total $P/2$ pares de indivíduos através deste método.

Cada par de indivíduos selecionados no operador de seleção é submetido a uma operação de cruzamento, que irá gerar dois indivíduos novos, chamados descendentes ($d_1$ e $d_2$). O operador de *crossover* aqui utilizado é denominado LSEC (*Local Search Enhanced Crossover*), proposto por Vallada e Ruiz (2011). Trata-se de uma variação do *One Point Order Crossover* para máquinas paralelas. No LSEC, para cada máquina $i$ do primeiro pai

($Parent_1$) é gerado um ponto $p_i$, escolhido aleatoriamente, com probabilidade uniforme entre $[1, |M_i|]$, sendo $|M_i|$ o número de tarefas presentes na máquina $i$. As tarefas presentes na máquina $i$ no intervalo $[1, p_i]$ são copiadas para a máquina $i$ do primeiro descendente ($d_1$). Já as tarefas entre $[p_{i+1}, |M_i|]$ são copiadas para a máquina $i$ do segundo descendente ($d_2$). Com isso, todas as tarefas do pai $Parent_1$ ficam distribuídas entre os dois descendentes. Neste ponto, as tarefas do segundo pai ($Parent_2$) são selecionadas para ingressar em $d_1$ e $d_2$. É verificado, separadamente, para cada descendente $d_x$ ($x \in [1, 2]$), quais tarefas alocadas no pai $Parent_2$ não fazem parte da solução parcial de $d_x$. Para cada máquina $i$ é criada uma lista $\bar{N}_i$ e as tarefas presentes na máquina $i$ do pai $Parent_2$, que não fazem parte do descendente $d_x$, são inseridas em $\bar{N}_i$, na mesma ordem em que elas estão alocadas em $Parent_2$. Em seguida, cada lista $\bar{N}_i$ é percorrida ordenadamente e cada tarefa $j$ presente em $\bar{N}_i$ é inserida na máquina $i$ de $d_x$ na posição que acarretar o menor incremento do $C_i$. Assim, o operador LSEC promove uma busca local limitada. A Figura 1 mostra um exemplo da aplicação deste operador.

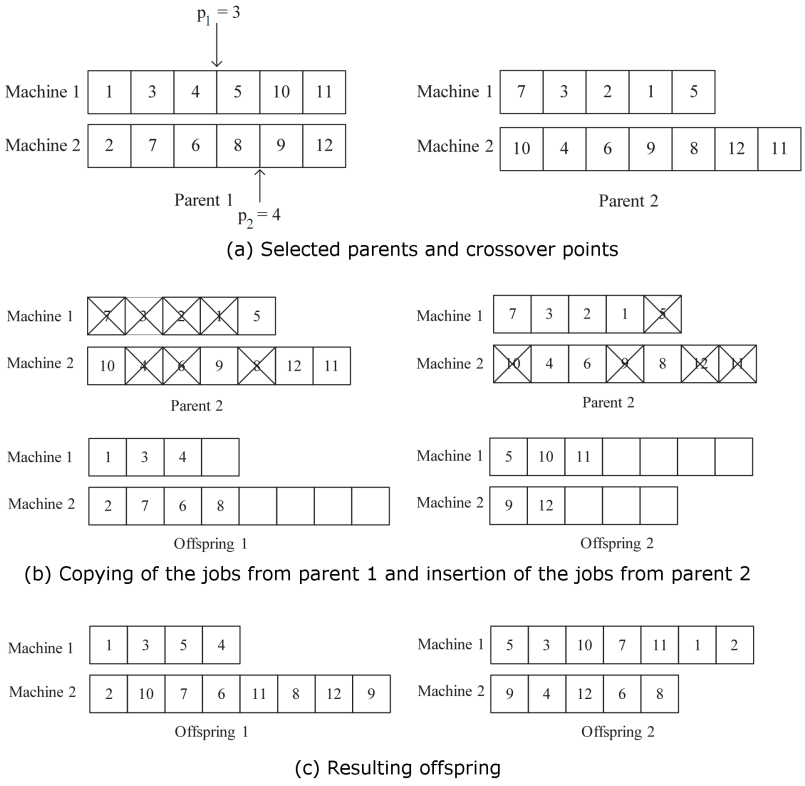



**Figura 1. Exemplo de operação de crossover LSEC. Fonte: Vallada e Ruiz (2011)**

Uma vez realizado o cruzamento, cada individuo é submetido, com probabilidade $prob_m$, ao processo de mutação. No presente trabalho, emprega-se o operador de mutação de deslocamento de tarefa, que, segundo Vallada e Ruiz (2011), é o mais aplicado em problemas de sequenciamento. De acordo com este operador, são escolhidas, de forma aleatória, uma máquina $i$, uma tarefa $j$ presente na máquina $i$ e uma posição $z$ da máquina $i$. Em seguida, a tarefa $j$ é deslocada para a posição $z$.

### 3.3 Busca Local - VND

A heurística *Variable Neighborhood Descent* (VND), proposta por Mladenovic e Hansen (1997), consiste na exploração cíclica de diferentes vizinhanças. A implementação do VND no presente trabalho envolve as vizinhanças de trocas internas, inserções externas e trocas externas, como mostra o pseudocódigo apresentado no Algoritmo 1.

**Algoritmo 1** Variable Neighborhood Descent

```
1: procedure VND(s)
2:     s* ← s;
3:     e ← 1;
4:     while e <= 3 do
5:         if e == 1 then
6:             s* ← explorar_vizinhanca_troca_interna(s);
7:         else if e == 2 then
8:             s* ← explorar_vizinhanca_insercao_externa(s);
9:         else
10:            s* ← explorar_vizinhanca_troca_externa(s);
11:        end if
12:        if makespan(s*) < makespan(s) then
13:            s ← s*;
14:        else
15:            e ← e + 1;
16:        end if
17:    end while
18:    return s;
19: end procedure
```

Em se tratando de minimização de *makespan*, os movimentos realizados devem, necessariamente, envolver a máquina $i$ com o maior instante de conclusão, isto é, ($C_i = C_{max}$), uma vez que o único meio de reduzir o *makespan* é por meio de alguma alteração na sequência desta máquina. Se utilizada esta característica do problema, o processo de busca local apresenta baixo custo computacional e alto desempenho.

#### 3.3.1 Exploração da vizinhança de trocas internas

Na exploração da vizinhança de trocas internas, são avaliadas as trocas entre cada par de tarefas $j$ e $k$ atribuídas à máquina $i$, $i \mid C_i = C_{max}$, realizando-se o movimento que promove a maior redução do *makespan*, se tal movimento existir.

#### 3.3.2 Exploração da vizinhança de inserções externas

As máquinas são ordenadas em uma lista $L$, segundo valores não-crescentes do instante de conclusão da última tarefa. Assim, a máquina $i$, tal que $C_i = C_{max}$, ocupa a primeira posição da lista. Seja $L^*$ a lista composta pelas máquinas $\{2, ..., m\}$ de $L$ e $w$ uma máquina em $L^*$, escolhida aleatoriamente, com probabilidade uniforme. A exploração da vizinhança de inserções externas consiste em avaliar a inserção de cada tarefa $j$ da máquina $i$ em cada uma das posições da máquina $w$, previamente escolhida. É realizado o movimento que promove a maior redução de *makespan*, se tal movimento existir, e os valores de $C_i$ e $C_w$ são, então, atualizados. Caso não exista um movimento que reduza o $C_{max}$, a máquina $w$ é retirada de $L^*$ e outra máquina é selecionada, de $L^*$, também aleatoriamente, reiniciando o processo. Este processo é repetido até que $L^*$ esteja vazia, ou até que alguma inserção reduza o valor de $C_{max}$.

### 3.3.3 Exploração da vizinhança de trocas externas

Considerando-se as listas $L$ e $L^*$, bem como as máquinas $i$ e $w$ definidas na seção anterior, cada uma das $j$ tarefas da máquina $i$ é trocada com cada uma das $k$ tarefas da máquina $w$. É realizado o movimento que promove a maior redução de *makespan*, se tal movimento existir, e os valores de $C_i$ e $C_w$ são atualizados. Caso não exista um movimento que reduza o $C_{max}$, a máquina $w$ é retirada de $L^*$ e outra máquina é selecionada de $L^*$, também aleatoriamente, reiniciando o processo, que é repetido até que $L^*$ esteja vazia, ou até que alguma troca reduza o valor de $C_{max}$.

## 3.4 Seleção da próxima população

Neste passo são definidos quais indivíduos que, após passarem pelos processos de seleção, cruzamento, mutação e busca local, irão ingressar na próxima população $P_{u+1}$. Diferentemente dos algoritmos genéticos convencionais, em que os indivíduos em $P_u$ são todos substituídos pelos indivíduos descendentes ($P_d$), neste trabalho apenas $X$ descendentes únicos, isto é, que não apresentam repetições em $P_u \cup P_d$ e que sejam melhores que os $X$ piores elementos da população anterior $P_u$, irão ingressar na população $P_{u+1}$. Os $P - X$ melhores indivíduos de $P_u$ completam o restante da população $P_{u+1}$.

# 4 Resultados Computacionais

Os experimentos realizados neste trabalho são conduzidos utilizando-se o conjunto de 1000 instâncias proposto por Vallada e Ruiz (2011) e disponível em SOA (2012). Este conjunto é composto por instâncias com $n = 50, 100, 150, 200, 250$ tarefas e $m = 10, 15, 20, 25, 30$ máquinas. Os tempos de processamento estão distribuídos uniformemente, entre 1 e 99. Em termos de tempos de preparação, são considerados 4 cenários, com diferentes intervalos de distribuição uniforme: $1-9$, $1-49$, $1-99$, $1-124$. Para cada uma das combinações de parâmetros (número de tarefas, número de máquinas e tempos de preparação), foram geradas 10 instâncias.

Os resultados alcançados por Vallada e Ruiz (2011) com o algoritmo genético (versão denominada GA2 por aqueles autores) são os melhores conhecidos até o presente momento para o conjunto de instâncias em questão e são tomados como referência para avaliação do desempenho das quatro versões do algoritmo avaliadas neste trabalho.

A primeira versão (GA2) implementada neste trabalho é a exatamente a apresentada em Vallada e Ruiz (2011). Nela, a população inicial é gerada de forma aleatória e um individuo gerado pela heurística MI (Kurz e Askin, 2001) é introduzido na população. Os operadores de seleção, cruzamento e mutação descritos na seção 3 são aplicados aos indivíduos e, em seguida, uma busca local na vizinhança de inserção externa é executado (maiores detalhes podem ser vistos em Vallada e Ruiz (2011)). Ao final, é executado o procedimento de seleção da próxima população, descrito na seção 3.4. Cabe observar que a busca local adotada em Vallada e Ruiz (2011) avalia movimentos entre máquinas que podem não ser as mais carregadas e, que, portanto, não têm a capacidade de reduzir o *makespan* da solução corrente.

A versão GA3 é a primeira versão proposta pelo presente trabalho e consiste na alteração da busca local do GA2. Propõe-se que sejam avaliados apenas os movimentos que retiram tarefas da máquina com maior *completion time*, efetivando-se o primeiro movimento de melhora. Na versão GA4, a busca local anterior é substituída pela exploração de

múltiplas vizinhanças, segundo o algoritmo VND, descrito na seção 3.3. Por fim, a versão GA5 diferencia-se da versão GA4 pela utilização da fase construtiva do GRASP (Seção 3.1) para a obtenção da população inicial.

Os parâmetros presentes nas quatro versões do algoritmo genético assumem os mesmos valores utilizados por Vallada e Ruiz (2011). A única exceção fica por conta do parâmetro $q$, que define o tamanho da lista restrita de candidatos (LRC) da fase construtiva do GRASP, presente na versão GA5. O valor de $q$ foi definido empiricamente, após a realização de extensivos experimentos computacionais. Sendo assim, os valores adotados para cada parâmetro foram: (i) tamanho da população: $|P_u| = 80$; (ii) tamanho da lista restrita de candidatos: $q = 0,05\,|\bar{N}|$; (iii) número de torneios para determinar cada par de indivíduos selecionados: $pressure = 20$; (iv) probabilidade de *crossover*: $prob_c = 0,5$; (v) probabilidade de mutação: $prob_m = 0,5$.

Todos os algoritmos implementados no presente trabalho foram desenvolvidos em linguagem Java, com JDK 1.6. Os experimentos foram executados em um ambiente com processador Intel i5-2450M com 2.5 GHz, 4GB de memória RAM e ambiente Windows 7 Home Premium. O critério de parada foi estabelecido em termos do tempo de execução, que é diretamente proporcional ao número de tarefas e ao número de máquinas da instância, de acordo com a expressão (Vallada e Ruiz, 2011) :

$$Time_i = n * (m/2) * 50(ms) \quad (1)$$

Cada um dos algoritmos (GA2, GA3, GA4 e GA5) foi aplicado 5 vezes a cada instância e, para cada resultado obtido, foi calculado o desvio percentual relativo (RPD):

$$RPD_{instance} = \frac{Method_{sol} - Best_{lit}}{Best_{lit}} \quad (2)$$

entre o *makespan* da solução encontrada por um dado algoritmo ($Method_{sol}$) e o *makespan* da melhor solução ($Best_{lit}$) obtida por Vallada e Ruiz (2011), disponíveis em SOA (2012).

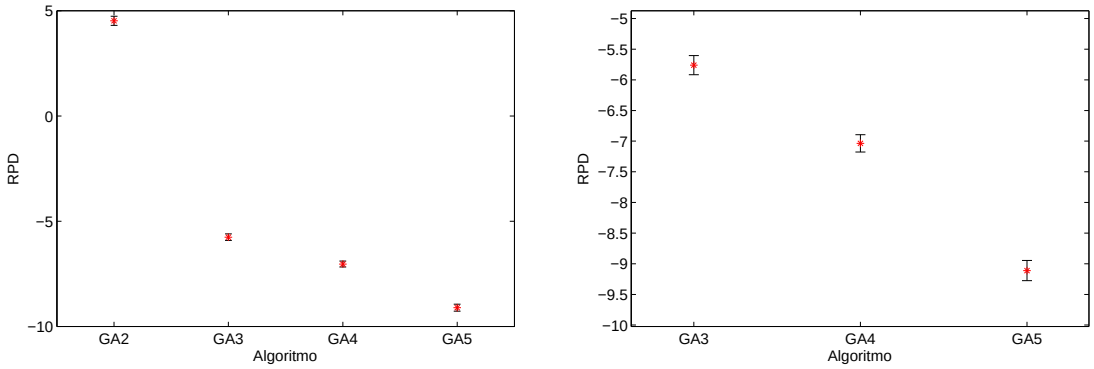
Em seguida, foram calculadas as médias dos desvios percentuais relativos para cada subconjunto de instâncias de mesmas dimensões, isto é, mesmos valores de $n$ e $m$. Como bem apontado por Vallada e Ruiz (2011), análises fundamentadas unicamente em valores médios, muito comuns na literatura, podem levar a conclusões errôneas, sendo recomendável uma análise mais refinada, a fim de verificar se as diferenças eventualmente observadas em tais valores médios têm significância estatística. Assim, utilizando-se o método estatístico Tukey HSD Montgomery (2001), são determinados os intervalos de confiança para as médias dos desvios percentuais relativos.

O conjunto de instâncias se mostra interessante e desafiador, uma vez que o modelo matemático proposto por Vallada e Ruiz (2011) não é capaz de resolvê-lo até a otimalidade para nenhuma instância em tempo computacional aceitável. Na Tabela 1 são apresentadas as médias dos desvios percentuais relativos. Os menores valores de desvios alcançados com a implementação do algoritmo genético de Vallada e Ruiz (2011) (GA2/New), em comparação aos desvios relatados em Vallada e Ruiz (2011) (GA2/Vallada), são explicados pelo maior poder de processamento do computador usado no presente trabalho. Na Tabela 1 pode ser observado que todas as variações propostas (GA3, GA4 e GA5) superam o algoritmo genético GA2/New, produzindo menores médias de desvios percentuais relativos. A superioridade das versões GA3, GA4 e GA5 é mais significativa para as instâncias com maiores números de máquinas. Vale lembrar que, na minimização de *makespan* em

**Tabela 1. Média do RPD por grupo n x m**

| Group | T(s) | GA2 | | GA3 | GA4 | GA5 |
|---|---|---|---|---|---|---|
| | | Vallada | New | | | |
| 50x10 | 12,5 | 6,49% | 2,54% | -0,33% | -1,04% | -1,46% |
| 50x15 | 18,75 | 9,20% | 3,05% | -2,25% | -5,20% | -5,76% |
| 50x20 | 25 | 9,57% | 4,91% | -1,43% | -7,39% | -7,84% |
| 50x25 | 31,25 | 8,10% | 4,55% | -0,84% | -9,36% | -10,11% |
| 50x30 | 37,5 | 9,40% | 5,38% | 1,65% | -8,78% | -9,13% |
| 100x10 | 25 | 5,54% | 0,24% | -2,12% | -2,22% | -2,72% |
| 100x15 | 37,5 | 7,32% | -0,23% | -4,60% | -5,26% | -6,37% |
| 100x20 | 50 | 8,59% | 1,88% | -6,01% | -7,33% | -8,97% |
| 100x25 | 62,5 | 8,07% | 2,50% | -8,39% | -10,67% | -12,36% |
| 100x30 | 75 | 7,90% | 4,50% | -7,12% | -10,95% | -13,76% |
| 150x10 | 37,5 | 5,28% | -0,20% | -3,47% | -3,33% | -3,84% |
| 150x15 | 56,25 | 6,80% | 1,52% | -4,96% | -4,46% | -5,71% |
| 150x20 | 75 | 7,40% | 1,99% | -6,95% | -6,65% | -9,42% |
| 150x25 | 93,75 | 7,05% | 2,59% | -9,02% | -9,72% | -12,56% |
| 150x30 | 112,5 | 7,17% | 6,23% | -9,62% | -10,41% | -14,02% |
| 200x10 | 50 | 4,24% | 1,64% | -4,47% | -4,13% | -4,66% |
| 200x15 | 75 | 6,21% | 3,32% | -5,90% | -5,54% | -7,44% |
| 200x20 | 100 | 6,71% | 5,91% | -7,82% | -7,07% | -10,57% |
| 200x25 | 125 | 6,82% | 7,37% | -9,96% | -8,35% | -13,58% |
| 200x30 | 150 | 7,09% | 5,45% | -9,45% | -9,76% | -15,07% |
| 250x10 | 62,5 | 4,38% | 4,11% | -4,53% | -4,21% | -1,75% |
| 250x15 | 93,75 | 5,12% | 6,55% | -7,18% | -6,78% | -8,43% |
| 250x20 | 125 | 6,92% | 6,46% | -8,28% | -7,65% | -11,17% |
| 250x25 | 156,25 | 6,02% | 6,23% | -10,37% | -9,60% | -14,36% |
| 250x30 | 187,5 | 5,91% | 5,42% | -10,59% | -10,02% | -16,45% |
| Média | 75 | 6,93% | 3,76% | -5,76% | -7,03% | -9,10% |

máquinas paralelas, qualquer movimento que não envolva a máquina mais carregada não tem a capacidade de diminuir o *makespan* da solução corrente. A busca local proposta por Vallada e Ruiz (2011) ignora esta característica do problema e, assim, esforço computacional é desperdiçado, avaliando soluções vizinhas não promissoras. O esforço computacional desperdiçado é tanto maior quanto maior o número de máquinas da instância. Todas as variações propostas levam em conta esta característica particular da minimização de *makespan*, tendo esta um importante papel nos melhores desempenhos dos algoritmos propostos. Deve-se observar, ainda, que novos melhores valores foram alcançados pelas variações propostas, para diversas instâncias (RPD negativos).




(a) Todos os algoritmos (b) Variações propostas

**Figura 2. Média e intervalos HSD com nível de confiança de 95% para todo conjunto de instâncias.**

A determinação dos intervalos de confiança para os desvios de cada algoritmo rati-

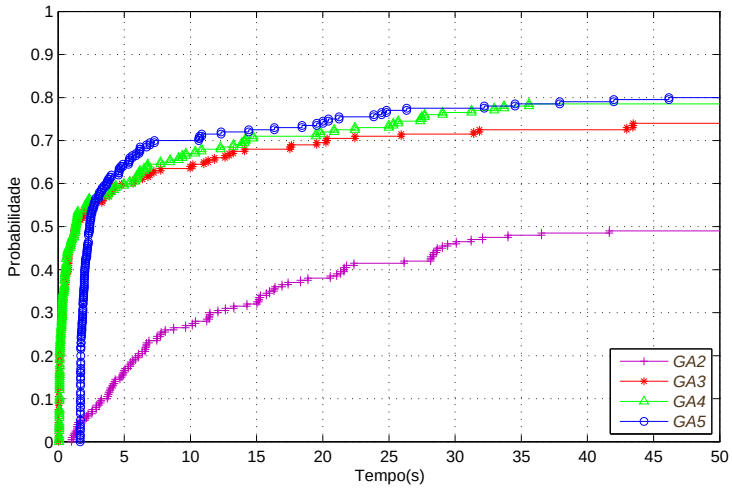



**Figura 3. Probabilidade para alcançar um alvo com desvio de 5% para melhor solução para instâncias de 100x10.**

fica, como pode ser observado na Figura 2(a), a superioridade das variações propostas sobre a versão original de Vallada e Ruiz (2011), quando aplicadas ao conjunto de instâncias em questão. Mais relevante, no entanto, é o que pode ser verificado na Figura 2(b), ou seja, a melhora de desempenho da versão GA4 sobre a versão GA3, proporcionada pela substituição da busca local de movimentos de inserção externa pela busca local de múltiplas vizinhanças (VND) e a melhora de desempenho da versão GA5 sobre a versão GA4, proporcionada pela utilização do GRASP na construção da população inicial. A eficiência dos algoritmos, medida em termos da probabilidade de se alcançar um alvo pré-estabelecido, em um intervalo de tempo fixo, é apresentada na Figura 3, para o conjunto de instâncias de 100 tarefas e 10 máquinas ($100 \times 10$). Na Figura 3, os alvos a serem atingidos consistem em soluções com *makespan* não superior a 5% em relação à melhor solução conhecida para cada instância (considerando não apenas a literatura, mas também os valores alcançados após todos os experimentos realizados neste trabalho). O intervalo de tempo foi fixado em 50 segundos (o dobro do tempo estabelecido para os experimentos anteriores) e cada algoritmo foi aplicado 5 vezes a cada instância. Observa-se mais uma vez o pior desempenho da versão GA2, que em apenas 50% das execuções consegue atingir os alvos especificados. Já a versão GA5 apresenta o desempenho inicial inferior às demais versões, devido ao maior custo para construir as soluções iniciais, mas, ao longo das execuções ultrapassa, as demais versões e chega ao alvo em 70% das execuções em menos de 10 segundos, e ao final dos 50 segundos consegue atingir o alvo em 80% das execuções.

## 5 Conclusão

Neste trabalho foram propostas melhorias no algoritmo genético proposto por Vallada e Ruiz (2011) para resolver o problema de minimização do *makespan* em máquinas paralelas não relacionadas, com tempos de preparação dependentes da máquina e da sequência de processamento das tarefas. Duas estratégias para geração da população inicial foram consideradas: (i) geração aleatória e (ii) geração parcialmente gulosa, via GRASP. No processo de busca local, são considerados apenas os movimentos que promovem alguma alteração na máquina que define o $C_{max}$, condição que deve ser observada para a diminuição do *makespan* da solução corrente. Assim, 4 versões do algoritmo foram estabelecidas e aplicadas

a um conjunto de 1000 instâncias disponíveis em SOA (2012). Os resultados mostram a superioridade média de 13% da versão GA5 proposta neste trabalho sobre o algoritmo genético proposto por Vallada e Ruiz (2011), com a obtenção de novas melhores soluções para diversas instâncias. Comparações entre as quatro versões do algoritmo genético apontam para melhorias de desempenho proporcionadas pelas estratégias de geração parcialmente gulosa da população inicial e de utilização da busca local com VND, sendo que todas as variações propostas neste trabalho ainda apresentam desempenho computacional muito superior à versão proposta por Vallada e Ruiz (2011).

## Agradecimentos

Agradecemos ao CNPq, ao CEFET-MG e à FAPEMIG e à CAPES pelo apoio ao desenvolvimento deste trabalho.

## Referências

Al-Salem, A. (2004). Scheduling to minimize makespan on unequal parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, v. 17, n. 1, p. 177–187.

Allahverdi, Ali e Gupta, J.N.D. (1999). A review of scheduling research involving setup considerations. *The international Journal of Management Science*, v. 27, p. 219–239.

Allahverdi, Ali; Ng, C.T.; Cheng, T.C.E. e Kovalyov, Mikhail Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, v. 187, p. 985–1032.

Arnaout, Jean Paul; Rabadi, Ghaith e Musa, Rami. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, v. 21, n. 1, p. 693–701.

Clarke, G. e Wright, J. R. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, v. 12, p. 568–581.

Coelho, I. M.; Souza, M. J. F. e Farias, R. (2012). A hybrid cpu-gpu local search heuristic for the unrelated parallel machine scheduling problem. *Third Workshop on Applications for Multi-Core Architecture*, v. , p. 19–23.

dePaula, Mateus Rocha; Ravetti, Martín Gómez; Mateus, Geraldo Robson e Pardalos, Panos 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, Thomas A. e Resende, Maurício G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 9, p. 849–859.

França, P. M.; Gendreau, M.; Laporte, G. e Muller, F. M. (1996). A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, v. 43, p. 79–89.

Gendreau, M.l; Laporte, G. e Guimarães, E. M. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 133, p. 183–189.

Guinet, A. (1991). Textile production systems: a succession of non-identical parallel processor shops. *Journal of the Operational Research Society*, v. 42/8, p. 655–671.

Helal, Rabadi M. e Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, v. 3, p. 182–192.

Kim, Dong-Won; Kim, Kyong-Hee; Jang, Wooseung e Chen, F. Frank. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, v. 18, p. 223–231.

Kim, Dong-Won; Na, Dong-Gil e Chen, F. Frank. (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. e Askin, R. G. (2001). Heuristic scheduling of parallel machines with sequence-dependent setup times. *International Journal of Production Research*, v. 39, p. 3747–3769.

Lancia, Giuseppe. (2000). Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *European Journal of Operational Research*, v. 120, p. 277–288.

Liaw, Ching-Fang; Lin, Yang-Kuei; Cheng, Chun-Yuan e Chen, Mingchin. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers and Operations Research*, v. 30, p. 1777?1789.

Manne, A. S. (1960). On the jobshop scheduling problem. *Operations Research*, v. 8, p. 219–230.

Martello, Silvano; Soumis, Franqois e Toth, Paolo. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete Applied Mathematics*, v. 75, p. 169–188.

Mladenovic, N. e Hansen, P. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, p. 1097–1100.

Montgomery, Douglas C. (2001). *Desing and Analysis of Experiments*. John Wiley and Sons, 5th edition edição.

Parker, R. G.; Deana, R. H. e Holmes, R. A. (1977). On the use of a vehicle routing algorithm for the parallel processor problem with sequence dependent changeover costs. *AIIE Transactions*, v. 9, p. 155–160.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 3th edition edição.

Rabadi, G.; Moraga, R. e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, v. 17, p. 85–97.

Rocha, Pedro Leite; Ravetti, Martín Gómez; Mateus, Geraldo Robson e Pardalos, Panos M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers and Operations Research*, v. 35, p. 1250–1264.

SOA,. Sistemas de optimización aplicada, accessed on October 20 2012, (2012). URL `http://soa.iti.es/problem-instances`.

Vallada, Eva e Ruiz, Rubén. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 211, p. 612–622.

Wagner, H. W. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistic Quarterly*, v. 6, p. 31–40.

Ying, K.; Lee, Z. e Lin, S. (2012). Makespan minimization for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*, v. 23, n. 5, p. 1795–1803.