X Congresso Brasileiro de Inteligência Computacional (CBIC'2011), 8 a 11 de Novembro de 2011, Fortaleza, Ceará

UM ALGORITMO BASEADO EM *ITERATED LOCAL SEARCH* PARA O PROBLEMA DE SEQUENCIAMENTO EM MÁQUINAS PARALELAS NÃO-RELACIONADAS COM TEMPOS DE PREPARAÇÃO DEPENDENTES DA SEQUÊNCIA

Matheus Nohra Haddad, Marcone Jamilson Freitas Souza
Haroldo Gambini Santos, Leandro Augusto de Araújo Silva
Departamento de Ciência da Computação
Universidade Federal de Ouro Preto
mathaddad@gmail.com, marcone@iceb.ufop.br, haroldo.santos@gmail.com, leandrogattuso88@gmail.com

Resumo – Este trabalho aborda o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. O objetivo é minimizar o tempo máximo de conclusão do sequenciamento, o chamado *makespan*. Visando sua resolução, é proposto um algoritmo heurístico, nomeado ILSVNDPR, que combina os procedimentos heurísticos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR). O procedimento VND examina o espaço de soluções por meio de trocas de estruturas de vizinhanças baseadas em movimentos de inserções e trocas de tarefas. Após a determinação de um ótimo local em relação às vizinhanças adotadas, aplica-se o procedimento PR com uma dada probabilidade, conectando esse ótimo local a uma solução elite formada durante o processo de busca. O algoritmo foi testado em conjuntos de problemas-teste da literatura e seus resultados comparados a uma versão do mesmo sem a aplicação do procedimento PR, assim como com dois algoritmos genéticos da literatura. Os experimentos computacionais mostraram que os resultados alcançados pelos algoritmos propostos superaram os resultados da literatura, em termos de qualidade da solução e variabilidade da solução. Também foram obtidos, para a maioria dos problemas-teste, novos limites superiores.

Palavras-chave – Sequenciamento em máquinas paralelas, Iterated Local Search, Variable Neighborhood Descent, Reconexão por Caminhos, *makespan*

Abstract – This paper addresses the unrelated parallel machine scheduling problem with sequence dependent setup times. The goal is to minimize the maximum completion of the schedule, called *makespan*. Aiming to its resolution, it is proposed a heuristic algorithm, named ILSVNDPR, which combines the heuristic procedures *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) and *Path relinking* (PR). The VND procedure examines the solution space through the exchange of neighborhood structures based on job insertions and swaps. After determining a local optimum in relation to the adopted neighborhood, it is applied the PR procedure with a given probability, connecting this local optimum to an elite solution created during the search process. The algorithm was tested on sets of test problems of the literature and the results compared to a version of the same procedure without the application of PR, as well as to two genetic algorithms from literature. The computational experiments showed that the results achieved by the proposed algorithms outperformed the results of the literature in terms of solution quality and variability of the solution. It were also obtained for most of the test problems, new upper bounds.

Keywords – Parallel machine scheduling, Iterated Local Search, Variable Neighborhood Descent, Path Relinking, *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 [1] e prática, pois existem muitas situações onde ele aparece, como por exemplo, em processos de fabricações em indústrias têxteis [2].

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. Em [3], os autores utilizaram uma Busca Tabu com o objetivo de minimizar o tempo total de atrasos com pesos. Em [4] são propostas sete heurísticas com o intuito de minimizar o tempo total de atrasos com pesos. Já em [5] é utilizado o método *Simulated Annealing* objetivando a minimização do tempo total de atraso. Um problema similar é tratado em [6], com datas de entrega iguais, sendo implementadas quatro heurísticas para solucioná-lo.

Um algoritmo genético adaptativo é proposto em [7] para o problema envolvendo datas de entregas, tendo como objetivo 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, [8] e [9]. 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 [9], a população inicial é criada com um indivíduo pela heurística de múltipla inserção [10] 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.

Este trabalho propõe o desenvolvimento de um algoritmo baseado na heurística *Iterated Local Search* – ILS [11], a fim de se obter soluções de melhor qualidade para o UPMSP tendo como critério de otimização o *makespan*. Na intenção de ampliar a exploração do espaço de soluções, foi utilizado o método *Variable Neighborhood Descent* – VND [12], responsável por realizar buscas locais em estruturas de vizinhanças. Além disso, também é incorporado ao algoritmo a aplicação da estratégia *Path Relinking* – PR [13] com o intuito de intensificar e diversificar a busca no espaço de soluções. Não só análises da aplicação da estratégia PR são feitas, para compreender sua importância, mas também comparações dos resultados alcançados com os encontrados na literatura.

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. Resultados computacionais são apresentados e discutidos na Seção 4. Finalmente, na Seção 5, 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 $R|S_{ijk}|C_{\max}$ [14].

Para melhor compreender o problema, seja um problema de sequenciamento envolvendo sete tarefas e duas máquinas. A Tabela 1 contém os tempos de processamento dessas tarefas nas duas máquinas, enquanto na Tabela 2 estão contidos os tempos de preparação dessas tarefas nessas máquinas. A Figura 1 ilustra um possível sequenciamento para esse exemplo.

Tabela 1: Tempos de processamento nas 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 nas 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

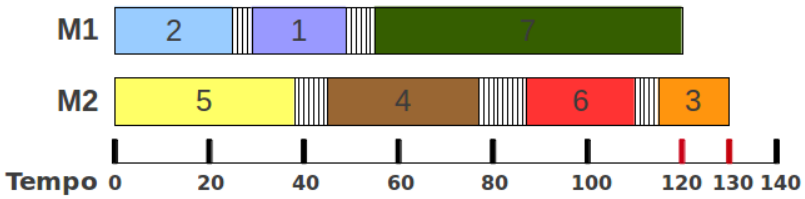

Figura 1: Exemplo de um possível sequenciamento

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 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 Algoritmo Proposto

Para resolver o UPMSP, propõe-se um algoritmo que combina os procedimentos heurísticos *Iterated Local Search* (ILS), *Variable Neighborhood Descent* (VND) e *Path Relinking* (PR). O VND é responsável por realizar as buscas locais do ILS e o PR é usado como estratégia de intensificação e diversificação da busca. O pseudocódigo deste algoritmo, nomeado ILSVNDPR, é mostrado no Algoritmo 1.

Algoritmo 1: ILSVNDPR

```
   Entrada: vezesnivel, criterioParada
1  Solucao s, s';
2  elite ← {};
3  s.geraSolucaoGulosa();
4  s.VND();
5  atualizaMelhor(s);
6  elite.insere(s);
7  nivel ← 1;
8  enquanto critério de parada não satisfeito faça
9      s' ← s;
10     vezes ← 0;
11     enquanto vezes < vezesnivel faça
12         perturbmax ← nivel + 1;
13         perturb ← 0;
14         s' ← s;
15         enquanto perturb < perturbmax faça
16             perturb ++;
17             s'.perturbacao();
18         fim
19         s'.avalia();
20         s'.VND();
21         elite.atualiza(s');
22         pr ← aleatorio(0,1);
23         se pr ≤ 0.1 ∧ elite.tamanho ≥ 5 então
24             el ← aleatorio(1,5);
25             PR(elite [el], s');
26         fim
27         se s'.fo < s.fo então
28             s = s';
29             atualizaMelhor(s);
30             elite.atualiza(s);
31         fim
32         vezes ++;
33     fim
34     nivel ++;
35     se nivel ≥ 4 então
36         nivel ← 1 ;
37     fim
38 fim
```

O Algoritmo 1 inicializa duas soluções na linha 1, bem como o conjunto de soluções elite (linha 2). A seguir, na linha 3,

é gerada uma solução gulosa, a qual passa por processos de buscas locais, por meio do procedimento VND (linha 4). Com o resultado destas buscas locais, a melhor solução conhecida até então é atualizada e, em seguida, essa solução é inserida no conjunto elite (linhas 5-6). O processo iterativo está situado nas linhas 8 a 38 e termina quando o critério de parada é satisfeito. Uma cópia da solução atual é feita na linha 9. O laço seguinte é responsável por controlar o número de vezes em cada nível de perturbação (linhas 11-33), sendo este número, $vezesnivel$, recebido como entrada do algoritmo. O próximo laço, linhas 15 a 18, executa as perturbações (linha 17), sendo que o número de vezes que o laço é executado depende do nível de perturbação. Com as perturbações realizadas, a solução obtida é avaliada na linha 19. Em seguida, é aplicado o procedimento VND nesta solução e é verificado se o ótimo local alcançado, em relação a todas vizinhanças adotadas no VND, pode ser inserido no conjunto elite (linhas 20 e 21). As linhas 22 a 26 controlam a aplicação do procedimento PR. Nas linhas (27-31) é verificado se as alterações feitas na solução contribuíram para uma melhor qualidade da mesma. As subseções seguintes apresentam detalhes do algoritmo proposto.

3.2 Representação da Solução

Uma solução s do UPMSP é representada como um vetor de listas. Em tal representação se tem um vetor v cujo tamanho é o número de máquinas, m, e cada posição desse vetor contém um número que representa uma máquina. O sequenciamento das tarefas em cada máquina, por sua vez, é representado por uma lista de números, em que cada número representa as tarefas. Para melhor compreensão desta representação tem-se como exemplo a Figura 2, na qual s representa o sequenciamento visto na Figura 1.

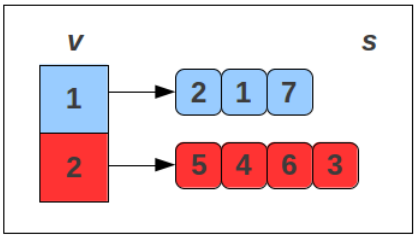

Figura 2: Representação como vetor de listas do UPMSP

3.3 Avaliação de uma Solução

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

3.4 Geração da Solução Inicial

A solução inicial é gerada pela aplicação da heurística *Adaptive Shortest Processing Time* (ASPT). Neste procedimento, primeiramente as tarefas são ordenadas em ordem crescente de tempos de processamento, ou seja, ordena-se de forma a se obter a máquina na qual as tarefas tem seu menor tempo de processamento. A tarefa de menor tempo de processamento é alocada na respectiva máquina. Para as outras tarefas, deve-se realizar a mesma ordenação descrita anteriormente, porém, nas máquinas em que já existem tarefas alocadas, deve-se somar a esse tempo, os tempos de preparação. Aloca-se a tarefa na máquina que tiver o menor tempo de conclusão. Esse processo de alocação termina quando todas as tarefas forem designadas.

3.5 Variable Neighborhood Descent

O procedimento *Variable Neighborhood Descent* (VND) tem como característica a exploração do espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhanças. Tais estruturas normalmente estão associadas a uma ordem pré-definida de aplicação. A cada iteração do VND é realizada uma busca local em uma estrutura de vizinhança. Ao fim dessa busca se tem o melhor vizinho da solução corrente em relação a esta estrutura de vizinhança. Se esse melhor vizinho não for melhor do que a solução corrente, então uma nova busca local na próxima estrutura de vizinhança é realizada. Caso esse melhor vizinho seja melhor do que a solução corrente, então a solução corrente passa a ser esse melhor vizinho e a busca local é reiniciada na primeira estrutura de vizinhança. O procedimento termina quando não é encontrada melhora em nenhuma das vizinhanças adotadas.

3.5.1 Estruturas de Vizinhanças

Este trabalho usa três estruturas de vizinhanças, tendo como base movimentos de trocas e inserções de tarefas. Essas estruturas serão descritas, a seguir, na mesma ordem em que são processadas pelo VND. A primeira vizinhança aplica movimentos de múltipla inserção, os quais são caracterizados por retirar uma tarefa de uma máquina e inserir em uma posição de outra máquina, inclusive a máquina à qual a tarefa já estava alocada. A busca na segunda vizinhança é realizada por movimentos de trocas de tarefas entre máquinas diferentes. A terceira e última vizinhança é analisada a partir de movimentos de trocas de tarefas na mesma máquina. São descritas, a seguir, as buscas locais implementadas com base nesses movimentos.

3.5.2 Buscas Locais

A primeira busca local utiliza 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. As remoções das tarefas são feitas começando nas máquinas com maiores tempos de conclusão e indo até as máquinas com menores tempos de conclusão. Já as inserções são feitas a partir das máquinas de menores tempos de conclusão até as máquinas de maiores tempos de conclusão. 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.

A segunda busca local realiza movimentos de trocas entre máquinas diferentes. Para cada par de máquinas existente são realizadas todas as trocas possíveis de tarefas entre elas. As trocas são feitas das máquinas que possuem maiores tempos de conclusão para as máquinas de menores tempos de conclusão. Os critérios de aceitação nessa busca são os mesmos aplicados na primeira busca. Se ocorrerem reduções dos tempos de conclusão nas duas máquinas envolvidas, então o movimento é aceito. Caso o valor reduzido do tempo de conclusão de uma máquina seja maior do que o tempo de conclusão acrescido da outra máquina, o movimento também é aceito. Assim que um movimento é aceito, a busca é interrompida.

A terceira busca local aplica movimentos de trocas na mesma máquina e utiliza a estratégia *First Improvement*. Para cada máquina são feitas todas as trocas possíveis entre suas tarefas. A ordem de escolha das máquinas é da máquina de maior valor de tempo de conclusão até a máquina que possui menor valor de tempo de conclusão. O movimento é aceito se o tempo de conclusão da máquina é reduzido. Conforme a aceitação do movimento, a busca é reiniciada e só termina quando for encontrado o ótimo local, em relação ao movimento de trocas na mesma máquina.

Avaliar uma solução totalmente a cada movimento de inserção ou troca é caro computacionalmente. Para tornar a busca local mais eficiente, é utilizado um procedimento que evita essa situação, avaliando apenas as máquinas que foram modificadas. Com isso, bastam algumas somas e diferenças para se obter o tempo de conclusão de cada máquina.

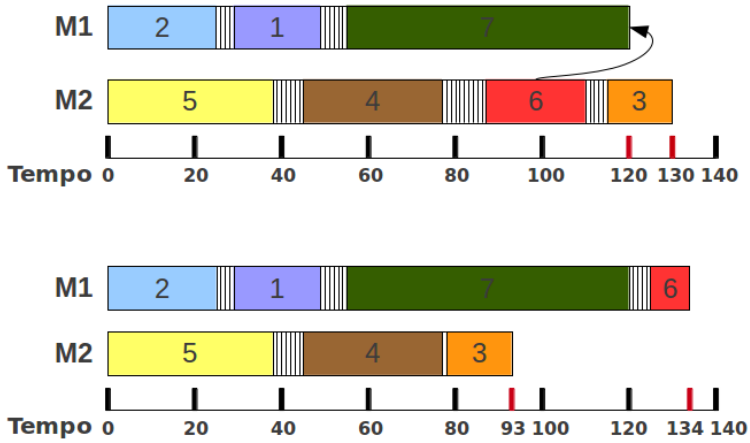

Figura 3: Avaliação no movimento de inserção

A Figura 3 ilustra esse procedimento de cálculo da função de avaliação, a partir de um movimento de inserção. Essa figura foi desenvolvida com base na Figura 2 e nos dados das tabelas 1 e 2. Percebe-se a retirada da tarefa 6 da máquina M2 e a sua inserção após a tarefa 7 da máquina M1. O procedimento de avaliação calcula o novo tempo de conclusão da máquina M2 subtraindo, deste, o tempo de processamento da tarefa 6, p_{26}, bem como subtraindo também os tempos de preparação envolvidos, s_{246} e s_{263}. É somado ao tempo de conclusão da máquina M2 o tempo de preparação s_{243}. Já na máquina M1, são somados ao seu tempo de conclusão, o tempo de processamento da tarefa 6 nesta máquina, p_{16}, e o tempo de preparação s_{176}. Como a tarefa 7 é a última a ser processada, nenhum tempo de preparação é necessário; assim, não é preciso subtrair nada. O novo tempo de conclusão da máquina M1 é calculado pela equação $M1 = 120 + 9 + 5 = 134$ e o novo tempo de conclusão da máquina M2 é calculado como $M2 = 130 - 23 - 10 - 5 + 1 = 93$.

Foi dado um exemplo da aplicação desse procedimento para avaliar um movimento de inserção. Ao tratar movimentos de trocas, a aplicação desse procedimento se torna trivial.

3.6 Perturbações

As perturbações consistem em aplicar movimentos de inserção a um ótimo local. Cada perturbação se caracteriza por retirar uma tarefa de uma máquina e inseri-la em outra máquina. Ao inserir a tarefa em outra máquina, procura-se a melhor posição para ela, ou seja, a tarefa será inserida na posição em que a máquina tiver o menor tempo de conclusão. A escolha das máquinas

e da tarefa é feita aleatoriamente. A quantidade de modificações em uma solução é controlada por um "nível" de perturbação, de forma que um determinado nível n de perturbação consiste na aplicação de $n + 1$ movimentos de inserção. O nível máximo permitido para as perturbações é 3; assim, ocorrerão 4 perturbações no máximo. O objetivo de se aumentar o nível de perturbação é diversificar a busca e procurar por uma solução de melhora em uma região gradativamente mais "distante" daquele ótimo local. O nível de perturbação somente é aumentado após geradas $vezesnivel$ soluções perturbadas sem que haja melhoria da solução corrente. Por outro lado, sempre que uma melhor solução é encontrada, a perturbação volta ao seu nível mais baixo. O procedimento utilizado na busca local para avaliar as soluções também é utilizado nas perturbações.

3.7 *Path Relinking*

A estratégia *Path Relinking* faz um balanço entre intensificação e diversificação da busca. Seu objetivo é explorar trajetórias que interligam soluções de alta qualidade. Para que isso seja feito, estas soluções de alta qualidade são armazenadas em um conjunto de soluções elite. Este conjunto possui duas regras para que uma solução faça parte dele. Uma solução é incluída no conjunto elite se ela tiver um menor valor de *makespan* do que o valor da melhor solução já presente no mesmo. Também é incluída no conjunto elite a solução que for melhor do que a pior solução do conjunto, mas que satisfaça a um nível mínimo de diversidade das outras soluções elite. O objetivo dessa segunda regra é a de evitar a inserção no conjunto elite de soluções muito parecidas. No algoritmo ILSVNDPR implementado o tamanho máximo do conjunto elite foi de 5 soluções e o nível mínimo de diversidade utilizado foi de 10%.

A diversidade entre duas soluções é definida como o percentual de tarefas diferentes nas mesmas posições. Para encontrar esse percentual de diversidade, uma comparação entre as tarefas a cada posição da solução é realizada. O somatório das posições que apresentarem tarefas diferentes é, então, dividido pelo total de posições da solução. O resultado desta divisão corresponde ao percentual de diversidade entre as soluções avaliadas.

De posse do conjunto elite, pode-se então construir os caminhos entre as soluções de alta qualidade, partindo de uma solução base e indo em direção a uma solução guia. Com esse fim, foi utilizada a estratégia regressiva (*backward*). Desta forma, considera-se a melhor solução como solução base e a pior solução como solução guia. No ILSVNDPR a solução base é escolhida, aleatoriamente, como uma das soluções presentes no conjunto elite e a solução guia é a solução resultante da aplicação do VND, isto é, um ótimo local. A cada iteração do PR, uma tarefa da solução base é realocada na mesma posição em que esta se encontra na solução guia. Feito isso, é realizada uma busca local com movimentos de múltipla inserção nesta nova solução. Este procedimento é repetido até que a solução base seja igual à solução guia. No ILSVNDPR, o PR é aplicado após a execução do VND, com probabilidade de 10% e somente quando se têm 5 soluções no conjunto elite.

4. RESULTADOS COMPUTACIONAIS

Para a realização dos testes computacionais, foi utilizado um conjunto de 360 problemas-teste da literatura, encontrados em [15], envolvendo combinações de 50, 100 e 150 tarefas e 10, 15 e 20 máquinas. Para cada combinação dessas existem 40 instâncias. Em [15] também são fornecidos os melhores valores de *makespan*, até então, para esses problemas-teste.

Foram implementados dois algoritmos, um utilizando o PR (ILSVNDPR) e o outro baseado somente no ILS com o VND (ILSVND). Ambos foram desenvolvidos na linguagem C++. Todos os experimentos foram executados em um computador com processador *Intel Core 2 Duo 2,4 GHz* com 2 GB de memória RAM e sistema operacional *Ubuntu 10.10*. O parâmetro utilizado nos algoritmos ILSVND e ILSVNDPR foi: *vezesnivel* $= 15$. O critério de parada, de ambos, foi o tempo máximo de duração do processamento $Tempo_{\max}$, em milissegundos, obtido pela Eq. (1), em que m representa o número de máquinas, n o número de tarefas e t é o parâmetro recebido como entrada para o Algoritmo 1. O parâmetro t foi testado com três valores para cada instância: 10, 30 e 50. Observa-se que esse critério de parada, com estes valores de t, foi o mesmo adotado em [9].

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

Na métrica utilizada para a comparação dos algoritmos, dada pela Eq. (2), o objetivo é verificar a variabilidade das soluções finais produzidas pelos algoritmos. Nesta 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. Assim como em [9], os algoritmos ILSVNDPR e ILSVND foram executados 5 vezes, para cada instância e para cada valor de t, calculando 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 os conjuntos de instâncias, o RPD_i^{avg} para cada valor de t dos algoritmos ILSVND, ILSVNDPR, bem como dos algoritmos GA1 e GA2 de [9]. Destaca-se que o GA1 e o GA2 foram executados em máquinas com a mesma configuração utilizada para os testes, *Intel Core 2 Duo 2,4 GHz* com 2 GB de memória RAM, possibilitando a comparação. Para cada conjunto de instâncias são encontrados três valores de RPD_i^{avg} separados por uma '/'. Esta separação representa resultados de testes alterando os valores de t, sendo a ordem $t = 10/30/50$ respeitada. Valores negativos indicam que os resultados encontrados superam os melhores de [9].

Tabela 3: Médias dos Desvios Percentuais Relativos dos algoritmos ILSVND, ILSVNDPR, GA1 e GA2 com $t = 10/30/50$

Instâncias	ILSVND	ILSVNDPR	GA1	GA2
50 x 10	3,01/2,24/2,07	**2,24/0,82/-0,20**	13,56/12,31/11,66	7,79/6,92/6,49
50 x 15	-0,47/-1,61/-2,33	**-2,57/-2,71/-2,50**	13,87/13,95/12,74	12,25/8,92/9,20
50 x 20	-1,46/-2,75/-3,41	**-2,23/-3,64/-4,47**	12,92/12,58/13,44	11,08/8,04/9,57
100 x 10	3,91/3,37/2,26	**2,79/1,31/0,95**	13,11/10,46/9,68	15,72/6,76/5,54
100 x 15	1,72/0,17/-0,35	**-0,24/-1,64/-2,03**	15,41/13,95/12,94	22,15/8,36/7,32
100 x 20	0,59/-1,84/-2,46	**-3,15/-3,36/-4,91**	15,34/13,65/13,60	22,02/9,79/8,59
150 x 10	3,19/1,74/1,26	**1,72/0,91/-0,01**	10,95/8,19/7,69	18,40/5,75/5,28
150 x 15	1,83/0,44/-0,34	**-0,17/-1,56/-1,91**	14,51/11,93/11,78	24,89/8,09/6,80
150 x 20	-0,27/-2,60/-3,24	**-2,92/-4,98/-5,82**	13,82/12,66/12,49	22,63/9,53/7,40
RPD^{avg}	1,34/-0,09/-0,73	**-0,50/-1,65/-2,32**	13,72/12,19/11,78	17,44/8,02/7,35

Na Tabela 3 estão destacados em negrito os melhores valores médios de RPD. Como se observa, o algoritmo ILSVNDPR é o que obteve os melhores resultados. Além de ganhar em todos os conjuntos de problemas-teste, ele ainda melhorou a maioria dos melhores resultados conhecidos até então. O algoritmo ILSVND também alcançou bons resultados, pois comparado com os algoritmos GA1 e GA2, ele também ganhou em todos os conjuntos de problemas-teste.

Para comprovar a vantagem do uso do mecanismo PR, foi realizado um teste de probabilidade empírica [16] com os algoritmos ILSVND e ILSVNDPR. Na execução dos experimentos, utilizou-se um problema-teste com 100 tarefas e 10 máquinas, tendo como alvo a melhor solução conhecida. Os dois algoritmos foram aplicados 100 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 $p_i = (i - 0,5)/100$. Os resultados do gráfico $t_i \times p_i$ são apresentados na Figura 4, na qual os tempos estão em milisegundos e em escala logarítmica.

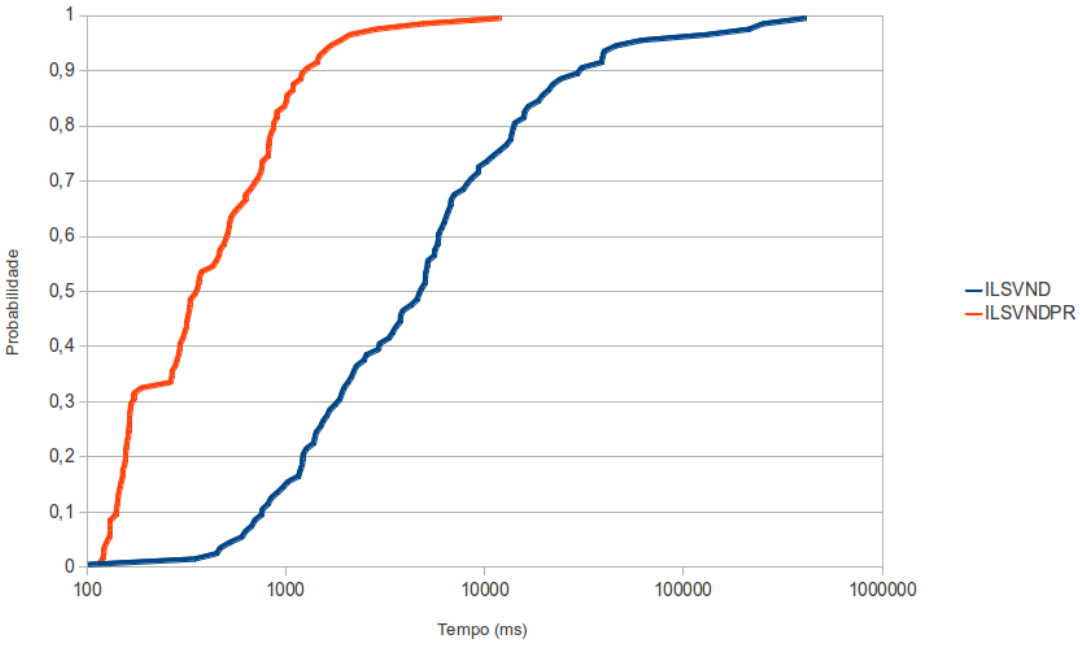

Figura 4: Teste de Probabilidade Empírica

Ao analisar a Figura 4 verifica-se a eficiência da aplicação do PR, pois com sua inclusão o valor alvo é alcançado por volta de 10 segundos, enquanto que no algoritmo ILSVND esse alvo somente é alcançado com quase 100% de probabilidade em cerca de 7 minutos.

5. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho abordou o problema de sequenciamento em máquinas paralelas não-relacionadas com tempos de preparação dependentes da sequência. O objetivo almejado foi a minimização do *makespan*.

Um algoritmo baseado em *Iterated Local Search* foi proposto para a resolução desse problema. Este algoritmo implementa o procedimento *Variable Neighborhood Descent*, como método de busca local. Também foi incluído no algoritmo acionamentos periódicos do procedimento *Path Relinking*.

Utilizando conjuntos de problemas-teste da literatura, o algoritmo proposto foi comparado a sua versão sem o procedimento PR, bem como com dois algoritmos genéticos da literatura. Os resultados computacionais mostraram que as duas versões do algoritmo são capazes de produzirem soluções melhores, com menores variabilidades e estabelecendo novos limites superiores. Todavia, pode-se notar a relevância do PR, pois com sua adição foram obtidos os melhores resultados e pelo teste de probabilidade empírica percebe-se melhora na eficiência, levando menos tempo pra atingir o valor alvo.

Como trabalhos futuros, o algoritmo será testado em todo o conjunto de problemas-teste disponível em [15], bem como será analisada a inclusão de uma fase de construção baseada no método *Greedy Randomized Search Procedure* (GRASP) [17] com o objetivo de partir de soluções iniciais de melhor qualidade.

6. AGRADECIMENTOS

Os autores agradecem às agências CNPq (processos 482765/2010-0 e 306458/2010-1) e FAPEMIG (processos CEX 00357/09 e CEX 01201/09) pelo suporte financeiro ao desenvolvimento desta pesquisa.

REFERÊNCIAS

[1] M. G. Ravetti, G. R. Mateus, P. L. Rocha and P. M. Pardalos. "A scheduling problem with unrelated parallel machines and sequence dependent setups". *International Journal of Operational Research*, vol. 2, pp. 380–399, 2007.

[2] M. J. Pereira Lopes and J. M. de Carvalho. "A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times". *European journal of operational research*, vol. 176, pp. 1508–1527, 2007.

[3] R. Logendran, B. McDonell and B. Smucker. "Scheduling unrelated parallel machines with sequence-dependent setups". *Computers & Operations research*, vol. 34, no. 11, pp. 3420–3438, 2007.

[4] M. X. Weng, J. Lu and H. Ren. "Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective". *International Journal of Production Economics*, vol. 70, pp. 215–226, 2001.

[5] D. W. Kim, K. H. Kim, W. Jang and F. Frank Chen. "Unrelated parallel machine scheduling with setup times using simulated annealing". *Robotics and Computer-Integrated Manufacturing*, vol. 18, pp. 223–231, 2002.

[6] D. W. Kim, D. G. Na and F. Frank Chen. "Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective". *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 173–181, 2003.

[7] P. A. Randall and M. E. Kurz. "Effectiveness of Adaptive Crossover Procedures for a Genetic Algorithm to Schedule Unrelated Parallel Machines with Setups". *Int. Journal of Operations Research*, vol. 4, pp. 1–10, 2007.

[8] M. R. de Paula, M. G. Ravetti, G. R. Mateus and P. M. Pardalos. "Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search". *IMA Journal of Management Mathematics*, vol. 18, pp. 101–115, 2007.

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

[10] M. Kurz and R. Askin. "Heuristic scheduling of parallel machines with sequence-dependent set-up times". *Int. Journal of Production Research*, vol. 39, pp. 3747–3769, 2001.

[11] H. R. Lourenço, O. Martin and T. Stützle. "Iterated Local Search". In *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, volume 57 of *International Series in Operations Research & Management Science*, pp. 321–353. Kluwer Academic Publishers, Norwell, MA, 2003.

[12] N. Mladenovic and P. Hansen. "Variable neighborhood search". *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[13] F. Glover. "Tabu search and adaptive memory programming - Advances, applications and challenges". In *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, edited by R. S. Barr, R. V. Helgason and J. L. Kennington, pp. 1–75. Kluwer Academic Publishers, 1996.

[14] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2008.

[15] 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.

[16] R. M. Aiex, M. G. C. Resende and C. C. Ribeiro. "Probability distribution of solution time in GRASP: An experimental investigation". *Journal of Heuristics*, vol. 8, pp. 343–373, 2002.

[17] T. Feo and M. Resende. "Greedy Randomized Search Procedure". *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.