Simulated Annealing e Iterated Local Search Adaptativos Aplicado ao Problema de Sequenciamento de Máquinas Paralelas não Relacionadas com Tempo de Preparação Dependendo da Sequencia

Lucas Uchôa Rodrigues [1], **Haroldo Gambine Santos** [1], **Túlio Ângelo Machado Toffolo** [1]

[1] Departamento de Computação - Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto (UFOP)
CEP: 35400-000 - Minas Gerais - MG - Brasil

choa.lucas@gmail.com, haroldo.santos@gmail.com, tulio@toffolo.com.br

Abstract. *This work addresses the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times. In this problem there is a set of jobs and machines and for each job there is a processing time associated, which is different for each machine. Given two jobs, there is also a setup time that depends on their sequence and on the machine used. The objective considered in this problem is to minimize the makespan. This NP-Hard problem was tackled in this work by Simulated Annealing and Iterated Local Search techniques, using several developed neighborhood structures. The parameters of the algorithms were set using the irace package. Computational experiments show that the proposed methods were able to outperform many results found in the literature, being able to improve the best known solutions for several instances.*
KEYWORDS *Scheduling, Simulated Annealing, Itarated Local Search*
Main Area Combinatorial Optimization, Metaheuristics

Resumo. *Este trabalho aborda o problema do Sequenciamento de Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependendo da Sequência. Nesse problema há um conjunto de máquinas e tarefas e para cada tarefa existe um tempo de processamento associado, que é diferente para cada máquina. Para cada par de tarefas existe também um tempo de preparação dependendo da sua sequência e da máquina para processa-las. O objetivo considerado neste problema é de minimizar o makespan de execução das tarefas. Este problema NP-Difícil foi abordado neste trabalho aplicando as técnicas Simulated Annealing e Iterated Local Search, usando diversas estruturas de vizinhanças. Os parâmetros para esses algoritmos foram ajustados utilizando o pacote IRace. Experimentos computacionais mostram que os métodos propostos foram capazes de superar muitos resultados encontrados na literatura, sendo capazes de melhorar as melhores soluções conhecidas para diversos casos.*
PALAVRAS CHAVE *Sequenciamento, Simulated Annealing, Itarated Local Search*
Área principal Otimização Combinatória, Metaheuristicas

1 Introdução

Esse trabalho lida com o problema do sequenciamento de máquinas paralelas não relacionadas com tempo de preparação dependente da sequência. Neste problema, existe um conjunto de tarefas e máquinas, no qual cada tarefa tem um tempo associado para execução em cada máquina. Entre duas tarefas, existe também um tempo de preparação, este depende da sequência das tarefas e da máquina associada. O objetivo é minimizar a quantidade de tempo necessária para conclusão da última tarefa em processamento (*makespan*).

Sendo o conjunto $J = j_1, ..., j_n$ o conjunto de n tarefas e $M = k_1, ..., k_m$ o conjunto de m máquinas, tal que:

1 : toda tarefa j deve ser executada apenas uma vez e apenas por uma única máquina;
2 : cada tarefa j tem um tempo de processamento p_{kj} se ela for executada pela máquina k;
3 : existe um tempo p_{kij} de preparação na máquina k para executar a tarefa j logo após a execução da tarefa i;
4 : existe um tempo p_{kii} para preparar a máquina k para executar a tarefa i se a tarefa i for a inicial.

O objetivo no problema é achar uma solução que aloque todas as tarefas J nas máquinas M tal que o tempo total de execução de todas as tarefas, chamado de $makespan$, seja o menor possível. A seguir um exemplo do problema, P. A Tabela 1 exibe o tempo de execução de seis tarefas em cada uma das máquinas do problema P. As linhas representam as tarefas e as colunas representam as máquinas.

Tabela 1. Tempo de execução das tarefas no exemplo do problema P

Tarefas	M1	M2
1	1	4
2	87	21
3	28	68
4	32	17
5	38	43
6	9	48

A Tabela 2 mostra o tempo de preparação das tarefas. Nessa tabela, os valores p_{kij} são apresentados. As linhas são os valores para i e as colunas os valores para j. A primeira célula nos mostra o valor de k. A diagonal mostra os valores para todo p_{kii}, o qual representa o tempo de preparação para i se a tarefa for a primeira a ser executada pela máquina. Nesse exemplo, $p_{kii} = 0, \forall k \in M, i \in J$

Tabela 2. Tempos de preparação do problema P

M1	1	2	3	4	5	6
1	0	1	8	1	3	9
2	4	0	7	3	7	8
3	7	3	0	2	3	5
4	3	8	3	0	5	2
5	8	3	7	9	0	5
6	8	8	1	2	2	0

M2	1	2	3	4	5	6
1	0	5	1	6	1	7
2	6	0	7	7	6	2
3	7	6	0	9	6	9
4	3	7	3	0	1	7
5	5	8	5	6	0	9
6	7	4	1	7	9	0

A Figura 1 mostra uma solução gulosa do exemplo P. Nessa solução, as tarefas 1, 6, 3 e 5, nessa ordem, são executadas pela máquina $M1$, enquanto as tarefas 4 e 2 são executadas pela máquina $M2$. A Figura 2 mostra a solução otimizada para o problema P. Quando comparadas as duas figuras, o makespan passa de 89 para 74 unidades de tempo, melhorando assim a solução em 15 unidades de tempo.

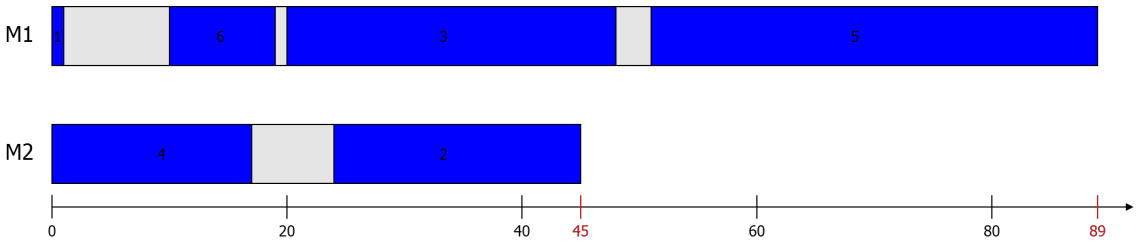

Figura 1. Exemplo de uma solução gulosa para o problema P

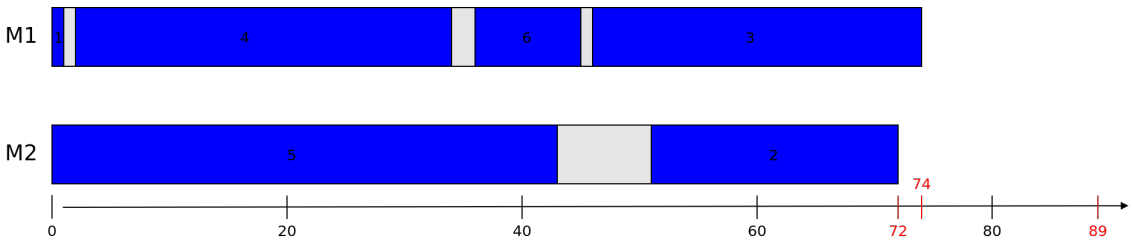

Figura 2. Solução otimizada do problema P

O problema geralmente é encontrado em industrias e pertence aos problemas da classe NP-Difícil [Glass (2001)]. Um levantamento completo dos problemas de programação com tempos de setup é apresentado em [Allahverdi (2008)]. O autor apresenta as principais diferenças e as várias abordagens algorítmicas para esta classe de problemas. O problema de programação de máquinas paralelas não relacionadas com a sequência de tempos de setup dependentes, também é o tema de [[Arnaout (2010)]-[Vallada (2011)]]. [Arnaout (2010)] implementou um algoritmo de busca adaptativo guloso e randômico [Feo (1995)] e também propôs algumas instâncias. Mais tarde, [Vallada (2011)] propôs um conjunto de instâncias para o problema (SOA-2011). Os autores implementaram um algoritmo genético [Goldberg (1989)], capaz de alcançar bons resultados dentro de uma pequena quantidade de tempo.

2 Método Construtivo Guloso

A fim de obter uma solução inicial viável, um algoritmo guloso simples foi desenvolvido. O algoritmo consiste na alocação de cada tarefa na melhor máquina, de acordo com o critério guloso apresentado aseguir. No começo, é criada uma lista com todas as tarefas que ainda não foram alocadas. Em seguida, cada tarefa desta lista é alocada, uma por uma, na extremidade da fila de execução da máquina que incorre no menor aumento para o makespan atual. O procedimento se encerra quando todas as tarefas são alocadas.

3 Estrutura das Vizinhanças

Para explorar o espaço de busca de diferentes soluções, foram desenvolvidos cinco estruturas de vizinhança. Essas vizinhanças são utilizados nas sub-rotinas das metaheurísticas implementadas durante a fase de busca local. As subseções a seguir detalham as estruturas de vizinhanças desenvolvidas.

3.1 Vizinhança Task Move

Um vizinho na estrutura de vizinhança Task Move é obtido movendo uma tarefa da máquina com maior tempo de execução total, Mx, para outra qualquer, My. A Figura 3.1 ilustra um exemplo da geração de uma vizinho utilizando Task Move.

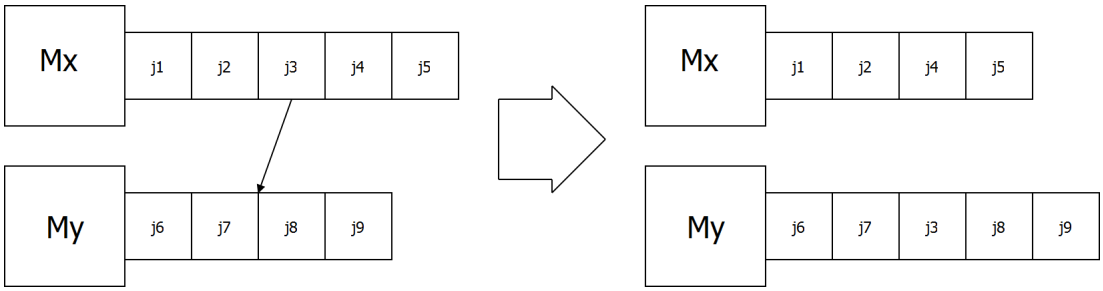

Figura 3. Exemplo de Vizinho na vizinhança TaskMove

3.2 Vizinhança Shift

Um vizinho na estrutura de vizinhança Shift é gerado por uma realocação de uma tarefa da máquina com maior tempo de execução total, Mx, em outra posição da mesma máquina. A Figura 3.2 ilustra um exemplo de como é gerado um vizinho utilizando Shift.

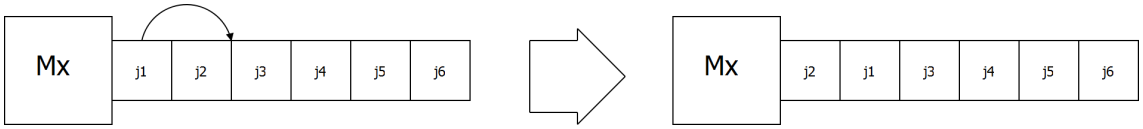

Figura 4. Exemplo de Vizinho na vizinhança Shift

3.3 Vizinhança Switch

Um vizinho na estrutura de vizinhança Switch é gerada trocando a ordem de duas tarefas na máquina com maior tempo de execução total, Mx. A Figura 3.3 nos mostra um exemplo dessa estrutura.

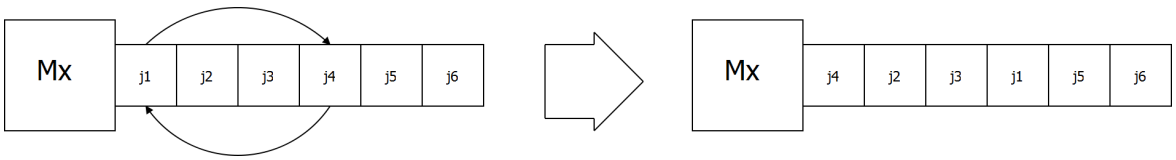

Figura 5. Exemplo de Vizinho na vizinhança Switch

3.4 Vizinhança Swap

Um vizinho na estrutura de vizinhança Swap é gerada pela troca de uma tarefa da máquina com o maior tempo de execução total, Mx, com uma tarefa de outra máquina, My. A Figura 3.4 apresenta um exemplo de um vizinho dessa estrutura. É importante notar que as posições das tarefas podem ser trocados colocado em qualquer posição sobre a outra máquina.

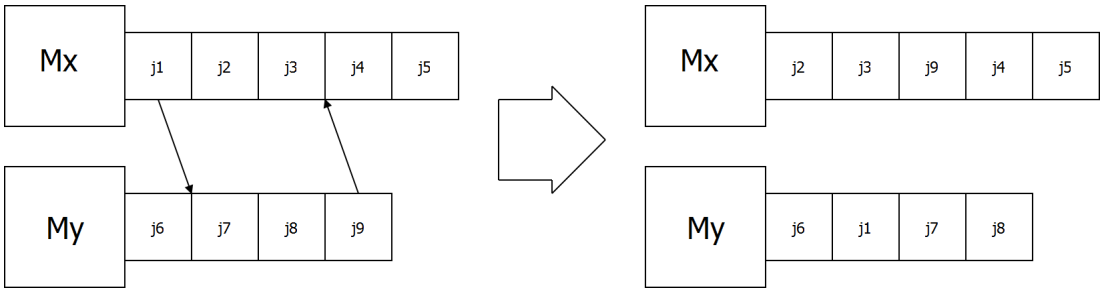

Figura 6. Exemplo de Vizinho na vizinhança Swap

3.5 Vizinhança 2-Realloc

Esta estrutura de vizinhança considera, de novo, a máquina com o maior tempo de execução total Mx. Para gerar um vizinho, duas tarefas que estão sendo executados pela máquina Mx tem sua ordem alteradas para posições aleatórias. Um exemplo é mostrado na Figura 3.5

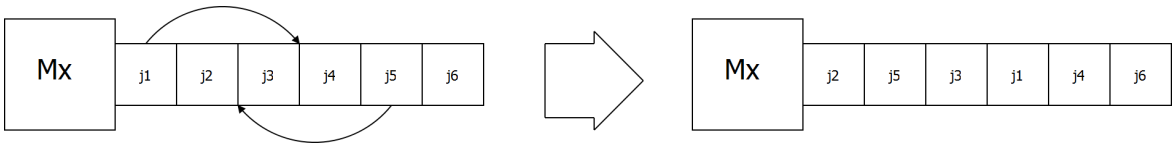

Figura 7. Exemplo de Vizinho na vizinhança 2-Realloc

4 Métodos Heurísticos

Duas heurísticas baseadas em busca local foram implementadas para este problema. O primeiro é o método Simulated Annealing, 1, utilizando diferentes estruturas de vizinhanças, e a segunda é método Iterated Local Search, ambos adaptativos. Um terceiro algoritmo, junta os dois métodos Simulated Annealing e o Iterated Local Search. As seções seguintes detalham a implementação desses métodos.

5 Simulated Annealing Adaptativo

Proposta em [Kirkpatrick (1983)], a metaheurística Simulated Annealing é um método probabilístico baseado na analogia com a termodinâmica que simula o arrefecimento de um conjunto de átomos aquecidos. Esta técnica começa a busca a partir de qualquer solução inicial. O procedimento principal consiste em um laço que gera aleatoriamente, em cada iteração, um vizinho s' da solução atual s. Seja Δ a variação do valor da função objetivo após mover-se para um vizinho ($\Delta = f(s') - f(s)$)O método aceita o movimento e o vizinho torna-se a nova solução corrente se $\Delta < 0$. Por outro lado, se

$\Delta \geq 0$ então o vizinho pode ser aceito com uma probabilidade de $e^{-\Delta/T}$, onde T é um parâmetro do método, chamado de temperatura que regula a probabilidade de aceitação de soluções piores.

A temperatura pode assumir, numa primeira fase, um valor T_0 elevado. Depois de um determinado número de iterações, SA_{MAX} (que representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico a uma dada temperatura), a temperatura é gradualmente reduzida por uma taxa α de resfriamento, de tal modo que $T_k \leftarrow \alpha * T_{k-1}$, e $0 < \alpha < 1$. Com este procedimento, uma maior possibilidade de evitar mínimo local ocorre na iteração inicial e, quando t se aproxima de zero, o algoritmo se comportar como um método de descida uma vez que reduz a probabilidade de aceitação de movimentos de piora. Algoritmo 1 mostra o pseudo-código do Simulated Annealing Adaptativo. Neste algoritmo, $f(.)$ É a função objetivo, $N_k(.)$ A estrutura de vizinhança k e o método $selecionaVizinhança(.)$ retorna a identificação de uma entre as 5 vizinhanças descrita na seção 3 aleatoriamente. O algoritmo tem os seguintes parâmetros:

s : uma solução inicial
T_0 : temperatura inicial
α : variável de resfriamento
SA_{max} : número de iterações em cada temperatura
$tempoLimite$: tempo limite

Algoritmo 1: Simulated Annealing

Entrada s, α, SA_{max}, T_0, $tempoLimite$
$s* \leftarrow s$
$T \leftarrow T_0$
$IterT :\leftarrow 0$
Ajusta uma mesma probabilidade para cada vizinhança de $selecionaVizinhança()$;
enquanto $tempoCorrente < tempoLimite$ **faça**
  enquanto $IterT < SA_{max}$ **faça**
    $IterT \leftarrow IterT + 1$;
    $k \leftarrow selecionaVizinhança()$; Gere um vizinho aleatório $s' \in N_k(s)$;
    $\Delta = f(s') - f(s)$;
    se $\Delta < 0$ **então**
      $s \leftarrow s'$;
      se $f(s') < f(s*)$ **então**
        $s* := s'$;
      fim
    senão
      faça $x \in [0, 1]$;
      se $x < e^{-\Delta/T}$ **então**
        $s := s'$;
      fim
    fim
  fim
  $T \leftarrow T * \alpha$;
  $IterT := 0$;
  se $T < 0.1$ **então**
    $T \leftarrow T_0$;
    Recalculando a probabilidade das vizinhanças de $selecionaVizinhança()$;
  fim
fim
Retorne $s*$

Uma das principais dificuldades do Simulated Annealing é o seu elevado número de parâmetros. O primeiro, T_0, é crítico, mas pode ser estimado automaticamente. [Ben-Ameur (2004)] aponta alguns métodos para calcular a temperatura inicial para se obter uma certa taxa de aceitação. Neste trabalho, usamos um método simples para calcular o valor de T_0. Dada uma solução inicial, s, um número dn de vizinhos diferentes são analisadas e o pior Δ obtido é armazenado. Este Δ é basicamente um valor para T_0 em que todos os vizinhos dn analisados seriam aceitos. Neste trabalho, uma taxa de 90% de aceitação é considerada, e assim o pior Δ é multiplicado por 0,9. O pseudo-código do método implementado é mostrado no Algoritmo 2. Este algoritmo tem três parâmetros: (i) a solução s inicial, (ii) o número dn de vizinhos a ser gerado e (iii) a $(taxa)$ da aceitação. Neste trabalho, todos os testes realizados utilizaram dn= 100 e a $taxa$= 0,9.

Algoritmo 2: Calculando T_0 do Simulated Annealing

```
1  Entrada s, dn, taxa
2  Δ ← 0;
3  i ← 0;
4  enquanto i < dn faça
5      k ← selecionaVizinhança();
6      Gera um vizinho aleatório s' ∈ N_k(s);
7      Δ_temp = f(s') − f(s);
8      se Δ_temp > δ então
9          Δ ← Δ_temp;
10     fim
11     i ← i + 1;
12 fim
13 Retorne Δ ∗ taxa;
```

6 Itaretad Local Search

O método Iterated Local Search (ILS) [Lourenco (2003)] baseia-se na ideia de que um procedimento de busca local pode conseguir melhores resultados, otimizando diferentes soluções geradas através de distúrbios na solução ótima local.

O algoritmo ILS começa a partir de uma solução inicial s_0 e faz perturbações de tamanho de p_{size} sob s_0, seguido por um método de descida. A perturbação é uma aceitação incondicional de um vizinho gerado por qualquer vizinhança apresentada na seção 3.

A fase de descida aceita os vizinhos se o seu valor de função objetivo é menor ou igual ao atual. Testado os movimentos são excluídos da vizinhança e retornados somente quando uma melhoria para a melhor solução de corrente for atingido. A fase de busca local termina quando não há nenhum vizinho remanescente a ser explorado. Existe um método de descida para cada tipo de estrutura de vizinhança apresentada na seção 3.

A busca local produz uma solução s' que será aceita se for melhor do que a melhor solução $s*$ encontrado. Em tal caso, o tamanho da perturbação p_{size} volta ao valor inicial p_0. Se a iteração Iter atinge o limite $Iter_{max}$, o tamanho da perturbação é incrementado. No entanto, se o tamanho da perturbação atinge um p_{max} limite, ele vai voltar para o p_0 inicial tamanho.

Algoritmo 3 apresenta o pseudo-código do ILS implementado. Neste algoritmo, $f(.)$ é a função objetivo, $N_k(.)$ a estrutura de vizinhança k, $selecionaVizinhança(.)$ um método que retorna aleatoriamente uma das vizinhanças descritas na seção 3 e um $metodoDaDescida$ representa a fase de descida. A fase de descida é uma busca de me-

lhorias em todas as vizinhanças, até que um ótimo local seja encontrado. Nesta pesquisa as vizinhanças são selecionadas em sequência e os vizinhos são enumerados. O algoritmo ILS tem os seguintes parâmetros:

s : solução inicial valida
p_0 : número de movimentos feito na perturbação inicial
p_{max}: número máximo de movimentos feito em uma perturbação
ils_{max} : número máximo de iterações
$tempoLimite$: o tempo limite de execução

Algoritmo 3: Iterated Local Search

Entrada $s, ILS_{max}, p_0, p_{max}, timeout$
$s \leftarrow metodoDaDescida(s)$;
$s* \leftarrow s$;
$p_{size} \leftarrow p_0$;
$iter \leftarrow 0$;
Ajusta uma mesma probabilidade para cada vizinhança de $selecionaVizinhança()$;
enquanto $tempoCorrente < tempoLimite$ **faça**
    $k \leftarrow selecionaVizinhança()$;
    $j \leftarrow 0$;
    enquanto $j < p_{size}$ **faça**
        Gera um vizinho $s' \in N_k(s)$;
    fim
    $s \leftarrow metodoDaDescida_k(s')$;
    se $f(s') < f(s*)$ **então**
        $s* \leftarrow s'$;
        $s \leftarrow s'$;
        $iter \leftarrow 0$;
        $p_{size} \leftarrow p_0$;
    senão
        $s \leftarrow s*$;
        $iter \leftarrow iter + 1$;
    fim
    se $iter \geq ILSmax$ **então**
        $p_{size} \leftarrow p_{size} + p_0$;
        se $p_{size} \geq p_{max}$ **então**
            $p_{size} \leftarrow p_0$;
            Recalculando a probabilidade das vizinhanças de $selecionaVizinhança()$;
        fim
    fim
fim
Retorne $s*$

7 Simulated Annealing e Iterated Local Search

Este procedimento é basicamente a execução do Iterated Local Search (ILS) após a execução do Simulated Annealing (SA). Este consiste em rodar o SA por um período de tempo SA_{time} e depois passar a melhor solução encontrada para o ILS. O ILS irá executar por $tempoTotal - SA_{time}$, aonde o $tempoTotal$ é o tempo total de execução do problema.

8 Formas Adaptativas

Para encontrarmos o melhor modo de se fazer uma perturbação ou uma busca local em uma solução utilizamos uma forma probabilística para escolha. A probabilidade de selecionar um tipo de vizinhança N_k é $p(\alpha_k)$, para $k = 1, 2, ..., m$. Os algoritmos SA e ILS

adaptativos atualizam os valores das probabilidades $p(\alpha_1), p(\alpha_2), ..., p(\alpha_m)$ para favorecer as vinhanças ou buscas locais que produzem as melhores soluções. Inicialmente as probabilidades são colocadas como $p(\alpha_i) = 1/m$; para $i = 1, 2, ..., m$, então, até o momento as vizinhanças são selecionadas uniformemente. Para atualizar, de forma adaptativa, as probabilidades, assumimos $F(S^*)$ como o valor da melhor solução encontrada até o momento e $I(\alpha_k) = 0$ para $k = 1, 2, ..., m$, o valor acumulado de melhora da melhor solução após ser aplicado a vinhança N_k ($F(S') - F(S^*)$). Periodicamente (nesse caso, a cada SA_{max} iterações) ocorre um reaquecimento do método e neste reaquecimento as probabilidades são atualizadas, de tal forma que $p(\alpha_k) = I(\alpha_k)/\sum I(\alpha_i)$, quando $I(\alpha_k)$ tem um valor diferente de zero, e $p(\alpha_k) = 0,1 * \sum I(\alpha_k)$, quando o valor de $I(\alpha_k)$ é igual a zero. Observe que quanto maior o valor de $I(\alpha_k)$ maior a probabilidade de se utilizar a vizinhança N_k, mas se esse valor for nulo, a vizinhança não será inutilizada. Após a atualização das probabilidades, o valor de $I(\alpha_k)$ é novamente zerado. O Mesmo procedimento é realizado para o ILS, sendo o número de iterações para atualização das probabilidade o valor ILS_{max}.

9 Experimentos

Nesta seção são detalhados os experimentos computacionais. As instancias de [Vallada (2011)] são utilizadas para avaliar os algoritmos implementados. Esse conjunto tem uma serie de valores diferentes para n e m. Os conjuntos utilizados tem os seguintes valores para n : 6, 8, 10, 12, 50, 100, 150, 200 e 250 e m: 2, 2, 4, 5, 10, 15, 20, 25 e 30.

Os algoritmos foram implementados utilizando C++ e o compilador C++ utilizado foi o 4.6.3. Todos os testes foram realizados em um computador com processador Intel Core i5 2.0 Ghz e 8 Gb de memória RAM, rodando Ubuntu12.04.

As instancias são nomeados de acordo com o seguinte padrão: $I - n - m - S - a - b - 1$. Nesse padrão I e S são irrelevantes e podem ser ignorados, n representa o número de tarefas, m o número de máquinas e, por fim, a e b representam o intervalo [a, b] dos possíveis valores para o tempo de execução e preparação para cada instancia, respectivamente.

Para calibrar os parâmetros do algoritmo, foi utilizado o pacote $iRace$. Esse pacote implementa um procedimento de corrida iterativa, que é uma extensão do Iterated F-race (I/F-race), proposto por [[Birattari (2006)],[Balaprakash (2007)]] e depois desenvolvido por [Yuan (2010)]. O objetivo principal do $iRace$ é fazer, de forma automática, a configuração dos algoritmos de otimização, ou seja, achar os parâmetros apropriados para os algoritmos. iRace é implementado como um pacote R ([R (2005)]), e baseia-se no pacote de corrida de [Birattari (2003)].

Após vários testes do iRace, os parâmetros escolhidos foram:

Simulated Annealing(SA): $SA_{max} \leftarrow 2850$ e $\alpha \leftarrow 0,95$
Iterated Local Search(ILS): $ILS_{max} \leftarrow 5245$ e $p_0 \leftarrow 1$
Execução Hibrida (SA+ILS): $SA_{max} \leftarrow 2650, \alpha \leftarrow 0,95, SA_{time} \leftarrow 26, ILS_{max}4190$ e $p_0 \leftarrow 1$, onde SA_{time} é o tempo que o método SA será executado. Em outras palavras, SA eve executar por 26 segundos e o ILS deverá executar por 34 segundos, sendo o tempo total dado 1 minuto.

A Tabela 3 representa os melhores resultados encontrados após 1 minuto de execução. Essa tabela mostra que 39 instâncias da literatura tiveram seus melhores re-

sultados conhecidos melhorados. Foram ocultados as instancias com 6, 8, 10 e 12 tarefas da tabela, pois em todos os casos os resultados foram iguais aos encontrados por [Vallada (2011)].

10 Conclusões e Trabalhos Futuros

Esse trabalho apresenta três heurísticas para o problema do Sequenciamento de Máquinas Paralelas Não Relacionadas com Tempo de Preparação Dependendo da Sequência. Os algoritmos propostos utilizam vários tipos de estrutura de vizinhanças e essas são capazes de produzir bons resultados. Considerando as instâncias de [Vallada (2011)], os métodos propostos foram capazes de melhorar a melhor solução conhecida na literatura em 39 dos casos após apenas 1 minuto de execução.

Para trabalhos futuros, será implementados mais tipos de estruturas de vizinhança, uma maior perturbação nas soluções no método Iterated Local Search e também testar outros tipos de metaheurísticas de busca local.

Tabela 3. Resultados

Results						
Instances	SA	ILS	SA + ILS	Nosso Melhor	Melhor Resultado SOA	Ocorreu Melhora?
I-50-10-S-1-124	133	121	120	120	114	
I-50-10-S-1-9	69	70	67	67	67	
I-50-15-S-1-124	93	77	84	77	75	
I-50-15-S-1-9	35	34	34	34	36	sim
I-50-20-S-1-124	62	56	54	54	52	
I-50-20-S-1-9	30	33	29	29	31	sim
I-50-25-S-1-124	42	33	40	33	37	sim
I-50-25-S-1-9	22	22	22	22	22	
I-50-30-S-1-124	33	29	27	27	27	
I-50-30-S-1-9	11	11	11	11	11	
I-100-10-S-1-124	280	237	238	237	221	
I-100-10-S-1-9	121	117	121	117	131	sim
I-100-15-S-1-124	183	143	143	143	141	
I-100-15-S-1-9	68	66	65	65	68	sim
I-100-20-S-1-124	129	96	110	96	100	sim
I-100-20-S-1-9	46	43	44	43	46	sim
I-100-25-S-1-124	99	74	80	74	73	
I-100-25-S-1-9	31	28	29	28	31	sim
I-100-30-S-1-124	76	56	65	56	61	sim
I-100-30-S-1-9	29	27	26	26	29	sim
I-150-10-S-1-124	408	336	339	336	332	
I-150-10-S-1-9	191	186	187	186	184	
I-150-15-S-1-124	258	194	205	194	210	sim
I-150-15-S-1-9	99	91	96	91	97	sim
I-150-20-S-1-124	189	130	143	130	141	sim
I-150-20-S-1-9	76	68	72	68	75	sim
I-150-25-S-1-124	149	108	111	108	112	sim
I-150-25-S-1-9	49	43	45	43	48	sim
I-150-30-S-1-124	120	84	86	84	85	sim
I-150-30-S-1-9	37	34	35	34	36	sim
I-200-10-S-1-124	512	402	405	402	417	sim
I-200-10-S-1-9	269	251	259	251	260	sim
I-200-15-S-1-124	316	241	244	241	253	sim
I-200-15-S-1-9	136	125	132	125	131	sim
I-200-20-S-1-124	234	172	178	172	176	sim
I-200-20-S-1-9	83	74	79	74	78	sim
I-200-25-S-1-124	182	136	137	136	141	sim
I-200-25-S-1-9	63	56	60	56	61	sim
I-200-30-S-1-124	140	104	103	103	122	sim
I-200-30-S-1-9	51	44	48	44	49	sim
I-250-10-S-1-124	605	490	483	483	500	sim
I-250-10-S-1-9	299	274	288	274	278	sim
I-250-15-S-1-124	350	284	292	284	313	sim
I-250-15-S-1-9	159	145	149	145	150	sim
I-250-20-S-1-124	275	220	216	216	232	sim
I-250-20-S-1-9	106	94	99	94	100	sim
I-250-25-S-1-124	213	160	161	160	175	sim
I-250-25-S-1-9	80	69	73	69	76	sim
I-250-30-S-1-124	158	128	129	128	138	sim
I-250-30-S-1-9	62	50	55	50	57	sim

Referências

Allahverdi, A., Ng, C. T., Cheng, T. C. E., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Arnaout, J.-P., Rabadi, G., and Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *J. Intell. Manuf.*, 21(6):693–701.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the f-race algorithm: sampling design and iterative refinement. In *Proceedings of the 4th international conference on Hybrid metaheuristics*, HM'07, pages 108–122, Berlin, Heidelberg. Springer-Verlag.

Ben-Ameur, W. (2004). Computing the initial temperature of simulated annealing. *Computational Optimization and Applications*, 29(3):369–385.

Birattari, M. (2003). The race package for R. Racing methods for the selection of the best. Technical Report TR/IRIDIA/
-2003-037, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Birattari, M., Balaprakash, P., and Dorigo, M. (2006). The aco/f-race algorithm for combinatorial optimization under uncertainty. In *Metaheuristics - Progress in Complex Systems Optimization. Operations Research/Computer Science Interfaces Series*, pages 189–203. Springer.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Glass, C., Potts, C., and Strusevich, V. (2001). Scheduling batches with sequential job processing for two-machine flow and open shops. *INFORMS J. on Computing*, 13(2):120–137.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Lourenco, H., Martin, O., and Stutzle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research e Management Science*, pages 320–353. Springer US.

R (2005). Development Core Team, R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. online.

Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622.

Yuan, Z., Stutzle, T., and Birattari, M. (2010). Mads/f-race: Mesh adaptive direct search meets f-race. In GarcÃa-Pedrajas, N., Herrera, F., Fyfe, C., BenÃtez, J. M., and Ali, M., editors, *IEA/AIE (1)*, volume 6096 of *Lecture Notes in Computer Science*, pages 41–50. Springer.