# Hybrid Algorithm for the Multi-objective Permutation Flow Shop Problem

**Volmir Fiorini Júnior**[1], **Carolina P. Almeida**[1], **Sandra M. Venske**[1]

[1]Departamento de Ciência da Computação (DECOMP)
Universidade Estadual do Centro-Oeste (UNICENTRO)
Guarapuava – PR – Brasil

volmir1999@hotmail.com, {carol, ssvenske}@unicentro.br

***Abstract.*** *Evolutionary algorithms are a non-deterministic approach to solving optimization problems that cannot be solved in polynomial time, like the classic NP-Hard problems. The Permutation Flow Shop (PFS) is a combinatorial optimization problem related to the production environment. In the PFS, jobs must be processed by machines, maintaining the same processing flow on the considered machines. In this work, we adopt the multiobjective approach for the PFS, minimizing the makespan and total flowtime. NSGA-II with Tabu Search were considered in this hybrid approach and applied to 11 instances of PSF with different dimensions. We include an analysis on the use of prohibition rules in Tabu Search and its restrictiveness. The results were analyzed using the IGD and Empirical Attainment Functions metrics and compared with the canonical NSGA-II.*

***Resumo.*** *Os algoritmos evolucionários são uma abordagem não determinística para resolver problemas de otimização que não podem ser resolvidos em tempo polinomial, como problemas clássicos NP-Hard. O Flow Shop de Permutação (FSP) é um problema de otimização combinatória do ambiente de produção, em que tarefas devem ser processadas por máquinas, mantendo o mesmo fluxo de processamento. Neste trabalho a abordagem multiobjetivo foi utilizada para o FSP, tendo como objetivos de minimização o makespan e o total flowtime. Dois algoritmos híbridos compostos por NSGA-II com Busca Tabu foram considerados na abordagem e aplicados em 11 instâncias do FSP com diferentes dimensões. Uma análise foi feita sobre o uso de regras de proibição na Busca Tabu e sua restritividade. Os resultados foram analisados utilizando as métricas de qualidade IGD e Função de Conquista Empírica, comparando-os com o NSGA-II canônico.*

## 1. Introdução

Um algoritmo de otimização procura uma solução ideal, transformando iterativamente uma solução candidata atual em uma nova solução, esperando que ocorra uma melhora. Os métodos de otimização podem ser divididos em duas classes principais, com base no tipo de solução localizada. Os algoritmos de busca local usam apenas informações locais do espaço de busca em torno da solução atual para produzir uma nova solução ótimo local. Diz-se que algoritmos de busca global exploram todo o espaço de pesquisa, enquanto algoritmos de busca local exploram vizinhanças [Engelbrecht 2007]. Quando

os dois métodos de otimização, global e local, são utilizados em conjuntos diz-se que o algoritmo é híbrido. Um algoritmo genético híbrido também é conhecido por algoritmo memético [Ishibuchi et al. 2003].

O *Flow Shop* de Permutação (FSP) é um problema de otimização combinatória do ambiente de produção, em que $n$ tarefas (ou *jobs*) devem ser processadas por $m$ máquinas distintas, mantendo o mesmo fluxo de processamento de máquinas. Dessa forma, o problema consiste em encontrar uma solução dentre as $(n!)$ sequências de tarefas possíveis [Conway et al. 2003].

Diversos trabalhos na literatura tratam o FSP multiobjetivo utilizando diferentes abordagens com busca local. Da mesma forma, diferentes objetivos a serem minimizados são utilizados: *makespan*, *maximum tardiness*, *total tardiness*, *maximum flowtime* e *total flowtime*. Em [Armentano and Claudio 2004] é utilizado um algoritmo paralelo com Busca Tabu para otimizar os objetivos *makespan* e *maximum tardiness*. Em [Varadharajan and Rajendran 2005] os autores utilizam um algoritmo baseado em *Simulated Annealing* para a minimização do *makespan* e do *total flowtime*. [Arroyo and Armentano 2005] propõem uma abordagem com busca local para otimizar dois pares de objetivos: (i) *makespan* e *maximum tardiness*; e (ii) *makespan* e *total tardiness*. A proposta em [Blot et al. 2017] é o uso de ILS (*Iterated Local Search*) para otimizar *makespan* e *total flowtime*. Em [Ishibuchi et al. 2003] os autores usam uma estratégia *hill climbing* simples em duas frentes: primeiramente são testados dois objetivos (*makespan* e *maximum tardiness*) e, num segundo teste o FSP é considerado com três objetivos (adicionando *total flowtime*). Em [Arroyo and Pereira 2011] a busca local VNS (*Variable Neighborhood Search*) é utilizada para a minimização de dois e três objetivos simultaneamente: (i) *makespan* e *maximum tardiness*; e (ii) *makespan*, *maximum tardiness* e *total flowtime*.

O objetivo deste trabalho é abordar o FSP multiobjetivo utilizando um algoritmo de busca global, NSGA-II (*Non-dominated Sorting Genetic Algorithm-II*) [Deb et al. 2002] associado a um algoritmo de busca local, conhecido como Busca Tabu [Taillard 1991, Glover 1986]. Os objetivos de minimização considerados são o *makespan* e o *total flowtime*. A principal contribuição deste trabalho está na análise sobre o uso de regras de proibição na Busca Tabu e sua restritividade, na minimização simultânea do *makespan* e do *total flowtime*.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica do trabalho com conceitos sobre o problema abordado (2.1), o algoritmo NSGA-II (2.2) e a Busca Tabu (2.3). O algoritmo proposto é detalhado na Seção 3 e os resultados e discussões são apresentados na Seção 4. Finalmente, a Seção 5 contém as conclusões e apontamentos para trabalhos futuros.

## 2. Fundamentação teórica

### 2.1. Problema *Flow Shop* de Permutação Multiobjetivo

Um Problema de Otimização Multiobjetivo (POM) pode ser definido como [Coello et al. 2007]:

$$\text{Minimizar:} \quad \mathbf{f} = \{f_1(\mathbf{x}), \ldots, f_O(\mathbf{x})\} \tag{1}$$

tal que, $\mathbf{f}$ é um vetor com $O$ funções objetivo conflitantes, $\mathbf{x} = (x_1, \ldots, x_D)$ é o vetor de variáveis de decisão, sujeito as restrições de domínio. Variáveis do espaço de decisão que obedecem à restrições definidas em (1) formam o espaço das soluções factíveis e o mapeamento realizado pelas funções objetivo leva ao espaço objetivo. No contexto multiobjetivo, uma solução domina outra se for igual ou melhor em todos os objetivos e melhor que a outra em pelo menos um objetivo. O conjunto de todas as soluções factíveis que não são dominadas por qualquer outra solução é chamado de conjunto ótimo de Pareto e sua imagem no espaço objetivo é chamada de fronteira de Pareto.

O FSP é um problema de programação de produção no qual $n$ tarefas devem ser processadas por um conjunto de $m$ máquinas distintas, tendo o mesmo fluxo de processamento nas máquinas. Usualmente a solução do problema consiste em determinar uma sequência das tarefas dentre as $(n!)$ sequências possíveis, que é mantida para todas as máquinas (programação permutacional). O problema possui as seguintes características [Conway et al. 2003]:

a) É dado um conjunto com $n$ tarefas $J_1$, $J_2$, ..., $J_n$;
b) É dado um conjunto com $m$ máquinas $M_1$, $M_2$, ..., $M_m$;
c) Cada tarefa demanda $m$ operações, com cada operação representando o tempo de processamento de cada tarefa por máquina;
d) As tarefas seguem o mesmo fluxo de operações nas máquinas, isto é, para qualquer $j = 1, 2, ..., n$, a tarefa $J_j$ deve ser processada primeiro na máquina $M_1$, depois na máquina $M_2$, e assim por diante até a última máquina, no caso a máquina $M_m$;
e) Uma máquina pode processar somente uma operação de cada vez e, iniciada uma operação, ela deva ser processada até a sua conclusão;
f) O número de sequências distintas possíveis para realização das tarefas nas máquinas é grande, isto é, $O(n!)$.

Em uma abordagem multiobjetivo, as duas funções objetivo para minimização consideradas neste trabalho são o *makespan* e o *total flowtime*. A minimização da duração total da programação é chamada de *makespan* e representa o intervalo de tempo entre o início de execução da primeira tarefa na primeira máquina e o término de execução da última tarefa na última máquina. O *total flowtime* corresponde ao tempo total de processamento, ou seja, soma do tempo decorrido entre o início e fim da execução de todas as tarefas em todas as máquinas.

Expressando uma solução como um vetor $\Pi := (\pi_1, \pi_2, ..., \pi_n)$ sendo que $\pi_i$ representa a tarefa de ordem $i$ a ser processada por todas as máquinas, o *makespan* e o *total flowtime* podem ser definidos por $C_{max}(\Pi) = C_{\pi_n,m}(\Pi)$ e $TFT(\Pi) = \sum_{i=1}^{n} C_{\pi_i,m}(\Pi)$ respectivamente, sendo que $C_{\pi_i,m}(\Pi), (i = 1, ..., n)$ é o tempo de conclusão de cada tarefa $\pi_i$ na máquina $m$, que pode ser calculado com seguinte equação:

$$C_{\pi_i,j}(\Pi) := \max\{C_{\pi_i j-1}(\Pi), C_{\pi_{i-1} j}(\Pi)\} + p_{\pi_i j}, \quad j = 1, ..., m \tag{2}$$

sendo que $p_{\pi_i,j}$ é o tempo de processamento da tarefa $\pi_i$ na máquina $j$ e $C_{\pi_i,0} := C_{\emptyset j}(\Pi) := 0$ para todo $i$ e $j$. Ø se refere à sequência nula inicial.

A Figura 1 representa esquematicamente o cálculo do *makespan* e do *total flowtime*. Dessa forma, verifica-se que o *makespan* é o tempo de conclusão da última tarefa

na última máquina e o *total flowtime* agrega todos os tempos de término das tarefas, em relação a última máquina.

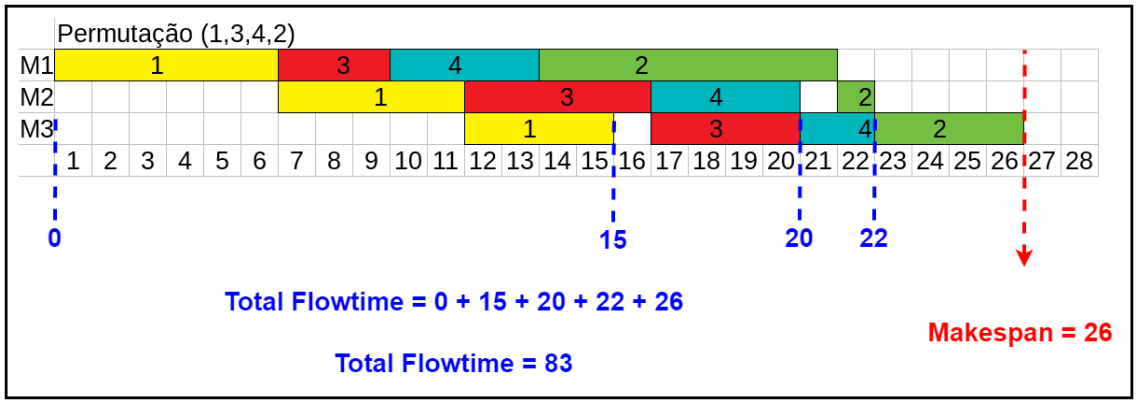



**Figura 1. Representação esquemática do cálculo das funções objetivo.**

Os valores de tempo de processamento de cada tarefa em cada máquina são dados de entrada do algoritmo, por meio das instâncias de teste. Os valores utilizados na Figura 1 foram obtidos pela matriz de entrada apresentada na Tabela 1. Neste exemplo, como existem 4 tarefas, o número de possibilidades é $n!$, ou seja, 24 permutações possíveis.

**Tabela 1. Exemplo de tabela de entrada do algoritmo contento o tempo de processamento de cada tarefa em cada uma das máquinas.**

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ |
|---|---|---|---|---|
| $M_1$ | 6 | 8 | 3 | 4 |
| $M_2$ | 5 | 1 | 5 | 4 |
| $M_3$ | 4 | 4 | 4 | 2 |

## 2.2. Otimização Multiobjetivo e NSGA-II

O NSGA-II é um algoritmo evolucionário multiobjetivo que funciona baseado em uma série de estratégias, juntamente com a ordenação das soluções ao longo do processo de busca, utilizando o conceito de dominância entre elas. Esse princípio de ordenar e manter as melhores soluções a cada geração contribui positivamente para a velocidade do algoritmo, além de prevenir que boas soluções sejam perdidas ao longo do processo evolutivo [Deb et al. 2002]. A aptidão de uma solução é calculada de acordo com o nível de dominância e sua distância em relação às outras soluções do mesmo nível, a chamada distância de multidão.

O funcionamento do algoritmo NSGA-II é apresentado na Figura 2, conforme sugerido por [Deb et al. 2002]. Inicialmente, com uma população gerada ($R_t$), é aplicada a ordenação não-dominada. Esta ordenação busca encontrar e separar as aproximações da fronteira de Pareto em vários índices ($F_1, F_2, F_3, ...$). Após encontrar a primeira fronteira de Pareto, esta recebe o índice 1 e é desconsiderada desse conjunto, para assim ser possível encontrar a próxima fronteira de Pareto de índice 2. Este processo continua subsequentemente até não ser possível criar mais grupos [Deb et al. 2002, Gaspar-Cunha et al. 2012].

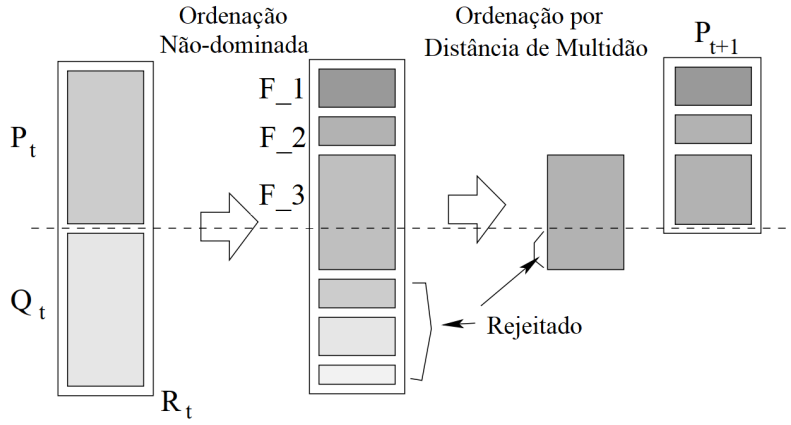



**Figura 2. Esquema do laço principal do NSGA-II. Fonte: [Deb et al. 2002].**

Com os grupos criados, metade da população precisa ser descartada, ou seja, os grupos de menores índices são mantidos e os de maiores são descartados. Porém, normalmente existirá um grupo que não poderá ser mantido integralmente nas soluções não descartadas, sendo necessário ocorrer uma seleção de parte deste grupo. Essa seleção é feita utilizando o conceito de Distância de Multidão [Deb et al. 2002, Gaspar-Cunha et al. 2012]. Após este processo, a população da próxima geração está selecionada ($P_{t+1}$). Na próxima geração são aplicados os operadores de mutação, cruzamento e seleção, mantendo a permutação e a factibilidade da solução [Deb et al. 2002].

### 2.3. Busca Tabu

A Busca Tabu (BT) [Glover 1986] é uma metaheurística projetada para escapar de um ótimo local, permitindo movimentos flexíveis. A BT seleciona um novo movimento de busca de forma que proíba temporariamente a avaliação de soluções anteriores. A BT básica é composto pelos seguintes elementos: i) Lista Tabu: este é o instrumento que representa uma memória para o algoritmo. A lista armazena e desativa os movimentos de buscas anteriores. Esses movimentos desativados são chamados de movimentos de tabu; ii) Período de locação: esse é o tempo, ou seja, o número de iterações em que a lista de tabu desativa os movimentos de tabu.

As regras de restrição definem a forma que a lista tabu funciona, podendo ser mais ou menos restritiva. Existem diversas regras de restrição na literatura [Gaspar-Cunha et al. 2012]. Neste trabalho, foram utilizadas as seguintes regras para movimentos de troca e inserção [Gaspar-Cunha et al. 2012]:

1. Regra $r_1$: impedir movimentos que resultem em uma permutação em que $i$ ocupe a posição $s_i$ ou $j$ ocupe a posição $s_j$.
2. Regra $r_2$: impedir que os elementos $i$ e $j$ troquem de posição com quaisquer outros elementos.

A regra $r_1$ é pouco restritiva e sua aplicação dá origem ao algoritmo denominado NSGA-II_BT$_{r1}$, enquanto a regra $r_2$ é bastante restritiva e sua aplicação origina o algoritmo NSGA-II_BT$_{r2}$.

Uma lista tabu pode ser muito restritiva, por isso, o chamado critério de aspiração permite que um movimento considerado tabu possa ser realizado. Os critérios de

aspiração mais comuns são aceitar um movimento, mesmo que tabu, se ele melhorar o valor da função objetivo global ou realizar o movimento tabu mais antigo se todos os possíveis movimentos forem tabus.

O procedimento de busca tabu segue os seguintes passos básicos:

1. Selecione uma solução inicial $s_0 \in$ S. Inicialize a Lista Tabu $L_0 = \emptyset$ e selecione um tamanho de tabu da lista. Estabeleça $k = 0$;
2. Avalie os elementos da vizinhança da solução inicial V($s_k$) que não consideram movimentos da lista tabu ($L_k$) ou que atendam ao critério de aspiração;
3. Substitua $s_0$ pela melhor solução;
4. Atualize a lista tabu, $L_{k+1}$;
5. Pare se uma condição de término for atingida; caso contrário, $k = k + 1$ e retorne ao passo 2.
6. Retorne a melhor solução encontrada.

## 3. Algoritmo Proposto

O pseudocódigo do algoritmo proposto é mostrado no Algoritmo 1. A abordagem tem como base o algoritmo multiobjetivo NSGA-II e o método de Busca Tabu.

**Algorithm 1** Pseudocódigo do Algoritmo Proposto

**Entrada** FSP $\mathbf{f}(\mathbf{x})$ e a população inicial $P_0$ de tamanho N
**Saída** Aproximação da Fronteira de Pareto de $\mathbf{f}(\mathbf{x})$

```
1: t ← 1
2: enquanto !CritérioParada() faça
3:     se CritérioAplicaçãoLS() então
4:         R ← PrimeiraFronteira(P_t)
5:         c ← Aleatório(1, tamanho(R))
6:         P_t[c] ← BuscaTabu(P_t[c], númeroDeVizinhos, númeroDeRodadas)
7:     fim se
8:     Q_t ← P_t ⋃ RecombinaçãoeMutação(P_t)
9:     F_1, F_2, ··· ← OrdenaçãoNão-Dominada(Q_t)
10:    i ← 1
11:    enquanto |P_{t+1}| + |F_i| < N faça
12:        P_{t+1} ← P_{t+1} ⋃ F_i
13:        i ← i + 1
14:    fim enquanto
15:    t ← t + 1
16:    P_t ← SeleçãoPorDistanciadeMultidão(F_i)
17: fim enquanto
```

As entradas para o algoritmo são o problema ($\mathbf{f}(\mathbf{x})$) e a população inicial ($P_0$). Inicialmente, a busca local é aplicada em uma solução aleatória obtida da primeira fronteira se o critério de aplicação da busca local for satisfeito (passos 3 a 7), substituindo a solução original $c$ na população $P$. O critério de aplicação utilizado é o parâmetro de frequência de aplicação da busca local. Os passos seguintes acompanham o que é proposto no algoritmo NSGA-II canônico. As soluções são modificadas pelos operadores e mescladas na população auxiliar $Q_t$ (passo 8). Em seguida, as soluções são classificadas de acordo com

o mecanismo de classificação de soluções não-dominadas (passo 9). Então, as fronteiras são sequencialmente adicionadas ($F_1$, $F_2$, ...) para a próxima população $P_{t+1}$ (passo 12). Quando a população não suporta todas as soluções da última fronteira, a seleção de Distância de Multidão (passo 16) é executada para escolher as melhores soluções na fronteira a serem adicionadas. Todas as etapas são repetidas até que o critério de parada seja alcançado. Neste trabalho, o critério de parada é o número máximo de avaliações.

O pseudocódigo do algoritmo da Busca Tabu é chamado no Algoritmo 2. Ele é utilizado no passo 6 do Algoritmo 1.

**Algorithm 2** Pseudocódigo da Busca Tabu

**Entrada** Solução $c$ da população $P$ em que será aplicada a Busca Tabu, o número de vizinhos a serem considerados ($númeroDeVizinhos$) e o número de rodadas de aplicação da Busca Local ($númeroDeRodadas$).

**Saída** A melhor solução encontrada durante o processo da busca.

1: $melhorSolução \leftarrow P[c]$
2: $soluçãoAtual \leftarrow P[c]$
3: **enquanto** !CRITÉRIOPARADA() **faça**
4: $Q \leftarrow$ GERARXVIZINHOS($soluçãoAtual, númeroDeVizihos$)
5: $Q \leftarrow$ RANKEAR($Q$)
6: $i \leftarrow 1$
7: **enquanto** !CRITÉRIOASPIRAÇÃO() e ÉTABU($Q[i]$) **faça**
8: **se** $i =$ TAMANHO($Q$) **então**
9: Voltar ao passo 4.
10: **fim se**
11: $i \leftarrow i + 1$
12: **fim enquanto**
13: ATUALIZARLISTATABU($Q[i]$)
14: $soluçãoAtual \leftarrow Q[i]$
15: **se** $Q[i].domina(melhorSolução)$ **então**
16: $melhorSolução \leftarrow Q[i]$
17: **fim se**
18: **fim enquanto**

O algoritmo de Busca Tabu tem como entradas a solução base ($P[c]$) em que será aplicada inicialmente a busca, o número de vizinhos a serem considerados durante o processo ($númeroDeVizinhos$) e o número de aplicações da busca local ($númeroDeRodadas$). Inicialmente a solução inicial é guardada como melhor solução, garantindo que a saída não seja uma solução pior que a entrada. Os vizinhos da solução $P_c$ atual são gerados e ranqueados (passos 4 e 5). O ranqueamento é feito utilizando a dominância entre as soluções, dispondo-as em um vetor baseado no índice da fronteira de Pareto de cada uma. Com os vizinhos ranqueados, o melhor é selecionado e verificado se o mesmo não satisfaz o critério de aspiração e é tabu (passo 7), em caso afirmativo, o próximo deve ser pego até encontrar a melhor solução não tabu gerada ou uma solução que satisfaça o critério de aspiração ou gerar um novo conjunto de vizinhos (passos 7 ao 12). Tendo a melhor solução não tabu ou que satisfaz o critério de aspiração, a lista tabu deve ser atualizada com o movimento que gerou a solução (passos 13). Essa solução é comparada com a melhor solução armazenada, substituindo-a caso seja melhor (passos 15

a 17). O processo é repetido, procurando um vizinho da solução atual de cada iteração, até que o critério de parada seja alcançado.

Foram utilizadas 2 regras diferentes (definidas na Seção 2.3) para definir o critério de solução tabu. O funcionamento dos passos de verificar se a solução é tabu e atualizar a lista tabu com uma solução dependem da regra que está sendo utilizada. O critério de aspiração utilizado neste trabalho é selecionar a solução caso ela seja melhor do que a melhor solução encontrada até então. O critério de parada utilizado é o número de rodadas de aplicação da busca.

## 4. Resultados e Discussão

Para o desenvolvimento do algoritmo proposto foi utilizado o *framework* jMetal [Durillo and Nebro 2011]. Para os testes, foram utilizadas algumas das instâncias propostas por Taillard [Taillard 1993]. As instâncias são divididas em 12 grupos de 10, sendo que cada grupo possui um tamanho, variando o número de tarefas e máquinas. A menor instância possui 20 tarefas e 5 máquinas, enquanto a maior possui 500 tarefas e 20 máquinas. Intuitivamente, quanto maior a quantidade de tarefas e máquinas de uma instância, maior a complexidade e tempo necessário para executar o algoritmo. Para este trabalho, foi escolhida a última instância de cada tamanho, com exceção do último grupo, por não estarem disponíveis arquivos para teste multiobjetivo - 11 instâncias no total. Cada instância está nominada de acordo com seu número (NR) e a combinação de tarefas ($n$) e máquinas ($m$) (TaNR$_{n-m}$). Foram feitas 10 execuções dos algoritmos com cada uma das instâncias selecionadas. Os principais parâmetros estão descritos na Tabela 2.

**Tabela 2. Parâmetros utilizados no algoritmo proposto.**

| | **Valores** | **Descrição** |
|---|---|---|
| **NSGA-II** | | |
| $n$ | Variável | Número de tarefas da instância (dimensão). |
| NP | 100 | Tamanho da população. |
| MAX-EV | $n$ * 1000 * NP | Número máximo de avaliações. |
| Cruzamento | 2P | Cruzamento de dois pontos aleatórios. |
| Mutação | Troca | Mutação de troca. |
| Taxa de mutação | 1.0 | Probabilidade de mutação. |
| Taxa de Cruzamento | 0.9 | Probabilidade de cruzamento. |
| **Busca Tabu** | | |
| Frequência de aplicação | 100 | Frequência de aplicação da busca tabu. |
| Duração do Tabu | 10 | Tempo de duração (rodadas) da restrição. |
| Número de vizinhos | 20 | Número de vizinhos gerados em cada rodada. |
| Rodadas de busca | 200 | Quantidade de rodadas de aplicação da busca. |
| Critério de Aspiração | Melhor | Selecionar a solução se ela for a melhor encontrada até o momento. |
| Movimento para geração da vizinhança | Troca | Tipo de movimento que determina a vizinhança da solução. |

Nos experimentos, primeiro foi feita uma análise de configuração para definir uma versão padrão, testando duas implementações da abordagem híbrida proposta com diferentes regras de proibição. O objetivo foi determinar qual das regras de proibição é melhor para o teste considerado.

As versões testadas, NSGA-II_BT$_{r_1}$ e NSGA-II_BT$_{r_2}$, utilizam duas diferentes regras de proibição para a Busca Tabu. Seja uma permutação (solução) $s$ de $n$ elementos. Em um movimento de troca, um elemento $i$ da posição $s_i$ é permutado com o elemento $j$ da posição $s_j$.

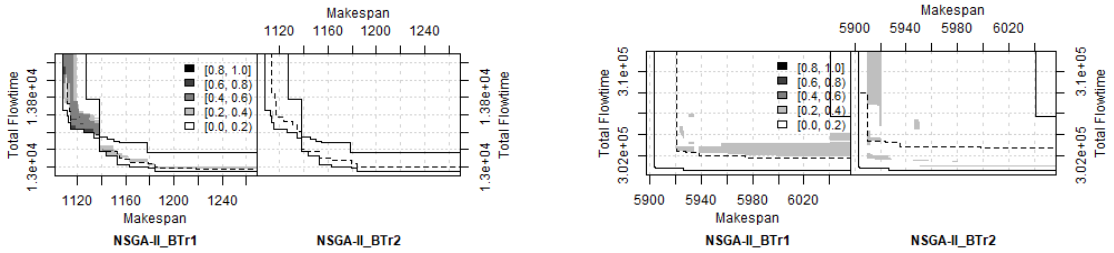
Os resultados são analisados com relação ao indicador de qualidade IGD (*Inverted Generational Distance*) e de Função de Conquista Empírica. Indicadores de qualidade são funções que atribuem um número real a um ou mais conjuntos de aproximações [Zitzler et al. 2008]. O IGD indica a que distância a fronteira de aproximação encontrada pelo algoritmo está de um conjunto de referência. O conjunto de referência é composto por todas as melhores soluções obtidas pelos algoritmos analisados. Valores mais baixos de IGD representam um melhor desempenho. Uma função de conquista empírica [Coello et al. 2007] é uma medida gráfica da eficiência de algoritmos evolucionários multiobjetivo. Ela define uma superfície que divide o espaço objetivo em metas (pontos), como por exemplo a fronteira de Pareto ótima, e a probabilidade de atingir cada meta. Pode-se utilizar esta métrica para identificar em quais regiões do espaço objetivo um conjunto de aproximação é melhor que o outro e qual a probabilidade disto acontecer. A função de conquista é uma métrica mais robusta que as demais, porém possui um custo computacional elevado [Coello et al. 2007]. As duas métricas foram adotadas por serem Pareto concordantes, ou seja, elas estabelecem uma relação de ordem coerente com o que é estabelecido pela dominância de Pareto [Zitzler et al. 2008].

A Tabela 3 apresenta os resultados comparativos entre as versões do NSGA-II com Busca Tabu utilizando separadamente as duas regras consideradas. A métrica utilizada foi o IGD. A versão da Busca Tabu com a regra $r_1$ obteve melhor desempenho quando aplicada às instâncias menores e a versão com a regra $r_2$ se destacou nas instâncias maiores (com mais de 100 tarefas e 20 máquinas). A Figura 3 ilustra a função de conquista empírica entre as duas regras para duas instâncias. Para a instância Ta010 (Figura 3 (a)) a regra $r_1$ é superior à $r_2$ no objetivo *makespan*. Para a instância Ta080 (Figura 3 (b)) a regra $r_1$ supera a regra $r_2$ para o objetivo do *total flowtime*.

**Tabela 3. Comparativo entre as duas regras de proibição para a Busca Tabu considerando o IGD (média e desvio padrão).**

| | NSGA-II_BT$_{r_1}$ | NSGA-II_BT$_{r_2}$ |
|---|---|---|
| Ta010$_{20-5}$ | $4.45e-03_{2.9e-03}$ | $8.06e-03_{3.6e-03}$ |
| Ta020$_{20-10}$ | $6.91e-03_{2.7e-03}$ | $7.15e-03_{3.8e-03}$ |
| Ta030$_{20-20}$ | $7.05e-03_{4.5e-03}$ | $5.99e-03_{3.7e-03}$ |
| Ta040$_{50-5}$ | $1.91e-02_{8.5e-03}$ | $2.45e-02_{1.1e-02}$ |
| Ta050$_{50-10}$ | $1.43e-02_{7.2e-03}$ | $2.15e-02_{1.5e-02}$ |
| Ta060$_{50-20}$ | $3.96e-02_{1.7e-02}$ | $4.28e-02_{1.3e-02}$ |
| Ta070$_{100-5}$ | $3.40e-02_{8.6e-03}$ | $3.49e-02_{1.7e-02}$ |
| Ta080$_{100-10}$ | $4.48e-01_{3.6e-01}$ | $5.85e-01_{4.6e-01}$ |
| Ta090$_{100-20}$ | $3.08e-02_{1.5e-02}$ | $1.59e-02_{6.8e-03}$ |
| Ta100$_{200-10}$ | $6.89e-02_{4.3e-02}$ | $5.41e-02_{2.8e-02}$ |
| Ta110$_{200-20}$ | $9.35e-02_{6.5e-02}$ | $9.15e-02_{3.7e-02}$ |

Quando comparadas com o NSGA-II canônico (sem busca local) as versões com Busca Tabu em conjunto de uma das duas regras apresentaram resultados diversos. A Tabela 4 apresenta os resultados comparativos utilizando IGD entre a versão com a regra menos restritiva ($r_1$) da Busca Tabu e o NSGA-II. O NSGA-II_BT$_{r_1}$ supera o NSGA-II em 7 das 11 instâncias, principalmente quando consideradas as instâncias menores. A Figura 4 ilustra a função de conquista empírica entre o NSGA-II com a BT aplicando a regra $r_1$ e o NSGA-II base para duas instâncias. Para a instância Ta010 (Figura 4 (a)) o algoritmo NSGA-II_BT$_{r_1}$ é superior ao NSGA-II no objetivo *makespan*. Para a instância Ta080 (Figura 4 (b)) o NSGA-II_BT$_{r_1}$ supera o NSGA-II em grande parte do espaço de busca, principalmente quando o objetivo do *total Flowtime* é observado.

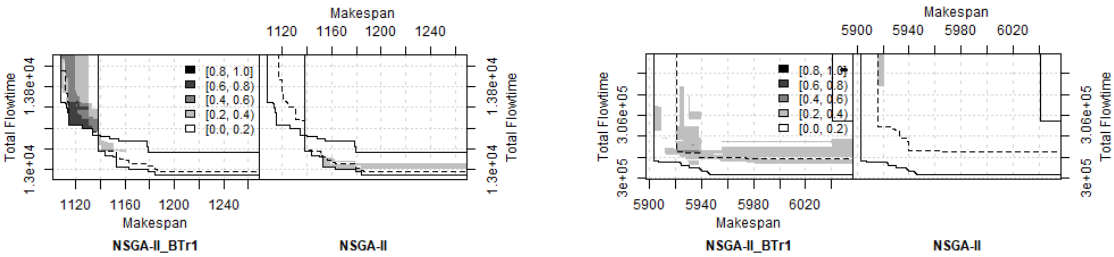



(a) Ta010 (b) Ta080

**Figura 3. Função de Conquista Empírica comparando o NSGA-II com as regras de proibição $r_1$ e $r_2$ para as instâncias Ta010 (a) e Ta080 (b).**

**Tabela 4. Comparativo da regra de proibição menos restritiva com NSGA-II considerando IGD (média e desvio padrão).**

| | NSGA-II_BT$_{r_1}$ | NSGA-II |
|---|---|---|
| Ta010$_{20-5}$ | $4.63e-03_{2.5e-03}$ | $7.80e-03_{4.3e-03}$ |
| Ta020$_{20-10}$ | $7.33e-03_{1.7e-03}$ | $7.78e-03_{3.8e-03}$ |
| Ta030$_{20-20}$ | $6.12e-03_{3.7e-03}$ | $3.89e-03_{2.3e-03}$ |
| Ta040$_{50-5}$ | $1.88e-02_{8.4e-03}$ | $2.17e-02_{6.9e-03}$ |
| Ta050$_{50-10}$ | $9.59e-03_{4.4e-03}$ | $1.09e-02_{3.8e-03}$ |
| Ta060$_{50-20}$ | $2.40e-02_{1.1e-02}$ | $2.55e-02_{7.2e-03}$ |
| Ta070$_{100-5}$ | $4.59e-02_{1.3e-02}$ | $3.06e-02_{1.7e-02}$ |
| Ta080$_{100-10}$ | $1.33e-01_{1.1e-01}$ | $1.79e-01_{1.2e-01}$ |
| Ta090$_{100-20}$ | $4.59e-02_{2.4e-02}$ | $3.67e-02_{2.8e-02}$ |
| Ta100$_{200-10}$ | $4.76e-02_{2.5e-02}$ | $5.10e-02_{2.5e-02}$ |
| Ta110$_{200-20}$ | $5.41e-02_{3.0e-02}$ | $4.01e-02_{2.1e-02}$ |



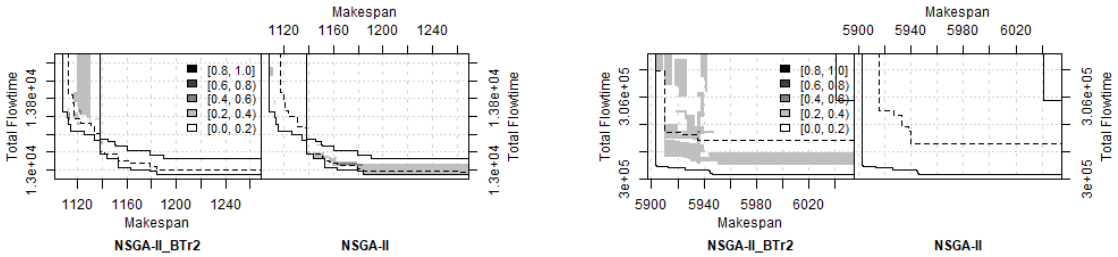

(a) Ta010 (b) Ta080

**Figura 4. Função de Conquista Empírica comparando NSGA-II_BT$_{r_1}$ e o NSGA-II para as instâncias Ta010 (a) e Ta080 (b).**

A Tabela 5 apresenta os resultados comparativos utilizando IGD entre a versão que emprega a regra mais restritiva ($r_2$) da Busca Tabu e o NSGA-II. O algoritmo híbrido, foi mais competitivo em comparação ao NSGA-II, principalmente considerando as instâncias de teste maiores. A Figura 5 ilustra a função de conquista empírica entre o NSGA-II com a BT aplicando a regra $r_2$ e o NSGA-II base para duas instâncias. Para a instância DD_Ta010 (Figura 5 (a)) o NSGA-II_BT$_{r_2}$ supera o NSGA-II para o objetivo do *makespan*, enquanto que para o *total Flowtime* o NSGA-II é superior. Para a instância Ta080 (Figura 5 (b)) NSGA-II_BT$_{r_2}$ é melhor que o NSGA-II em grande parte do espaço objetivo.

**Tabela 5. Comparativo da regra de proibição mais restritiva com NSGA-II considerando IGD (média e desvio padrão).**

| | NSGA-II_BT$_{r_2}$ | NSGA-II |
|---|---|---|
| Ta010$_{20-5}$ | $8.12e-03_{2.8e-03}$ | $8.77e-03_{4.4e-03}$ |
| Ta020$_{20-10}$ | $7.45e-03_{2.3e-03}$ | $7.67e-03_{3.5e-03}$ |
| Ta030$_{20-20}$ | $4.88e-03_{3.0e-03}$ | $3.65e-03_{2.4e-03}$ |
| Ta040$_{50-5}$ | $1.67e-02_{1.1e-02}$ | $1.51e-02_{6.3e-03}$ |
| Ta050$_{50-10}$ | $1.86e-02_{1.3e-02}$ | $1.39e-02_{5.1e-03}$ |
| Ta060$_{50-20}$ | $2.87e-02_{9.2e-03}$ | $2.89e-02_{1.0e-02}$ |
| Ta070$_{100-5}$ | $4.81e-02_{2.8e-02}$ | $2.87e-02_{1.9e-02}$ |
| Ta080$_{100-10}$ | $3.60e-01_{2.7e-01}$ | $3.91e-01_{2.4e-01}$ |
| Ta090$_{100-20}$ | $1.85e-02_{6.0e-03}$ | $2.77e-02_{1.6e-02}$ |
| Ta100$_{200-10}$ | $4.47e-02_{1.8e-02}$ | $5.95e-02_{2.9e-02}$ |
| Ta110$_{200-20}$ | $9.47e-02_{4.8e-02}$ | $6.07e-02_{4.0e-02}$ |




(a) Ta010 (b) Ta080

**Figura 5. Função de Conquista Empírica comparando NSGA-II_BT$_{r_2}$ e o NSGA-II para as instâncias Ta010 (a) e Ta080 (b).**

## 5. Conclusões

No presente trabalho foi apresentada a aplicação de busca local em uma plataforma multiobjetivo para a solução do *Flow Shop* de Permutação. A abordagem multiobjetivo envolveu a minimização do *makespan* e do *total flowtime* e teve por base o algoritmo NSGA-II. O mecanismo de busca local considerado foi a Busca Tabu e foi incorporado ao NSGA-II. Duas versões foram propostas, cada uma implementando uma regra de proibição diferente. Os experimentos foram conduzidos, inicialmente, para observar o comportamento do mecanismo de proibição. Neste sentido a regra menos restritiva, NSGA-II_BT$_{r_1}$, apresentou melhores resultados para as métricas de qualidade utilizadas.

Na sequência os testes avaliaram o efeito da busca local desenvolvida no algoritmo base, o NSGA-II. O resultados apontaram que o algoritmo NSGA-II_BT$_{r_1}$ é capaz de melhorar os resultados do NSGA-II em boa parte dos casos de teste considerados.

A busca local proposta apresenta diferentes parâmetros para balancear o esforço computacional com exploração do espaço de busca. Como trabalhos futuros, é possível considerar técnicas que controlem automaticamente estes parâmetros. Uma alternativa atraente é o uso de hiper-heurística para selecionar os parâmetros, inclusive ao longo da evolução.

## Referências

Armentano, V. A. and Claudio, J. E. (2004). An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10(5):463–481.

Arroyo, J. E. and Pereira, A. (2011). A grasp heuristic for the multi-objective permutation flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 55:741–753.

Arroyo, J. E. C. and Armentano, V. A. (2005). Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, 167(3):717 – 738. Multicriteria Scheduling.

Blot, A., Pernet, A., Jourdan, L., Kessaci-Marmion, M.-É., and Hoos, H. H. (2017). Automatically configuring multi-objective local search using multi-objective optimisation. In Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M., Jin, Y., and Grimme, C., editors, *Evolutionary Multi-Criterion Optimization*, pages 61–76, Cham. Springer International Publishing.

Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A., et al. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

Conway, R. W., Miller, L. W., and Maxwell, W. L. (2003). *Theory of scheduling*. Dover.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771.

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

Gaspar-Cunha, A., Takahashi, R., and Antunes, C. H. (2012). *Manual de computação evolutiva e metaheurística*. Imprensa da Universidade de Coimbra/Coimbra University Press.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549. Applications of Integer Programming.

Ishibuchi, H., Yoshida, T., and Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4):443 – 455.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Varadharajan, T. and Rajendran, C. (2005). A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research*, 167(3):772–795.

Zitzler, E., Knowles, J., and Thiele, L. (2008). Quality assessment of pareto set approximations. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, volume 5252 of *Lecture Notes in Computer Science*, pages 373–404. Springer Berlin Heidelberg.