ConBRepro
XI CONGRESSO BRASILEIRO DE ENGENHARIA DE PRODUÇÃO
EVENTO ON-LINE
01 a 03 de dezembro 2021

# Implementação de um algoritmo Hill-Climbing para o problema de Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos

**Ana Carolina Piccinini de Alencar Schiavi**
Engenharia de Produção - Universidade Tecnológica Federal do Paraná
**Arthur Kreling Ozorio**
Engenharia de Produção - Universidade Tecnológica Federal do Paraná
**Danilo Wakabayashi**
Engenharia de Produção - Universidade Tecnológica Federal do Paraná
**Pedro Henrique Terra da Silva**
Engenharia de Produção - Universidade Tecnológica Federal do Paraná
**Rafael Henrique Palma Lima**
Engenharia de Produção - Universidade Tecnológica Federal do Paraná

**Resumo:** Este trabalho tem como objetivo apresentar um algoritmo de busca na vizinhança para a resolução do problema de Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos (UPMSRC - *Uniform Parallel Machine Scheduling with Resource Constraints*) que possui como obstáculo a alocação de tarefas (*jobs)* nas máquinas de forma a minimizar o valor de *makespan* (menor utilização de recursos). Para isso, foi proposto um método para geração de soluções iniciais e uma estrutura de vizinhança para exploração de soluções candidatas. No estudo foram analisadas 32 instâncias por meio de um algoritmo criado em linguagem de programação *Python* que contém funções para a geração de uma solução inicial aleatória factível e execução do *Hill-Climbing*. O algoritmo realizou o procedimento de todas as instâncias em pouco mais de 34 minutos obtendo como resultado um GAP médio de 21,78% em relação às soluções obtidas a partir da resolução utilizando o Solver Gurobi, concluindo que o algoritmo é eficiente, porém ainda podem existir outros modelos com soluções melhores.

**Palavras-chave:** *Hill-Climbing*, *Python*, Pesquisa Operacional.

# Implementation of a Hill-Climbing algorithm for the Uniform Parallel Machine Scheduling problem with Resource Constraints (UPMSRC)

**Abstract:** This work aims to present a neighborhood search algorithm to solve the Sequencing problem in Uniform Parallel Machines Scheduling with Resource Constraints which has as an obstacle the allocation of tasks (jobs) in the machines in order to minimize the value of makespan (less use of resources). For this, a method for generating initial solutions and a neighborhood structure for exploring candidate solutions were proposed. In the study, 32 instances were

analyzed through an algorithm created in Python programming language that contains functions for the generation of a feasible random initial solution and Hill-Climbing execution. The algorithm performed the procedure for all instances in just over 34 minutes, resulting in an average GAP of 21.78% in relation to the solutions obtained from the resolution using Solver Gurobi, concluding that the algorithm is efficient, however there may still be other models with better solutions.

**Keywords:** Hill-Climbing, Python, Operacional Research.

## 1. Introdução

O planejamento e programação da produção se tornou algo imprescindível na atualidade, visto que o contato com clientes de todo o mundo trouxe uma grande demanda, levando as empresas a necessitar destas atualizações para se manterem competitivas no mercado. Algoritmos se tornaram grandes aliados desse cenário, uma vez que trazem como resultado sempre uma solução quase ótima para o problema de planejamento em questão.

O problema de Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos é um destes que são frequentes e importantes na produção e que podem ser resolvidos por meio de um algoritmo. Para que a solução ótima seja encontrada, será utilizado o método *Hill-Climbing*.

De acordo com Bennet (2004), o método *hill-climbing* é quando se começa por uma solução aleatória e, a cada passo, é feita uma mutação aleatória que é aceita desde que não diminua a aptidão da resposta. Com este modelo, a solução encontrada será quase ótima. O *Hill-Climbing* busca soluções a fim de decidir qual será o próximo passo baseado na análise das soluções existentes na vizinhança da solução atual. Dada uma solução, a estrutura de vizinhança é o conjunto de soluções que se encontram próximo a ela segundo algum critério e geralmente é encontrada fazendo apenas pequenas mudanças na estrutura da solução inicial (COLNAGO *et al*, 2019)

Este artigo apresenta a seguinte estrutura: descrição do problema, em que o problema de Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos será descrito e abordagens na literatura serão apresentadas; proposta do algoritmo, que contém uma descrição dos elementos do algoritmo conforme implementado; resultados obtidos, em que será explicada a forma como o algoritmo foi executado e os resultados serão apresentados; e, por fim, as conclusões.

## 2. Descrição do Problema (Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos)

Conforme Li *et al* (2019), o sequenciamento é uma função essencial na gestão da produção, além de, também, ser uma importante ferramenta de decisão no nível operacional para a companhia responder rapidamente às demandas dos clientes. Ele é importante em problemas com máquinas paralelas, uma vez que esse modelo é universal no ambiente da manufatura e é altamente usado em sistemas de informação ou em serviços, como transportadoras, bancos, etc. De acordo com Pinedo (2012), problemas de sequenciamento de produção lidam com a alocação de recursos para resolver tarefas em um dado intervalo de tempo e têm como objetivo otimizar algo. Eles podem ser encontrados em diversas áreas, como, por exemplo, no planejamento da produção, no gerenciamento de projetos, no sequenciamento de tarefas, no controle de decolagens e aterrissagens, entre outros (ABREU e PEREIRA, 2018).

Mundim e Fuchigami (2017) definem as máquinas paralelas como um conjunto de recursos ou máquinas que executam qualquer tarefa ou atividade, podendo ser classificadas em: idênticas (máquinas possuem velocidade igual para a mesma tarefa),

uniformes (velocidade da máquina não depende da tarefa e sim da máquina), ou não relacionadas (a velocidade de processamento depende da tarefa e da máquina). No presente trabalho é estudado o caso em que as máquinas paralelas são uniformes.

No sequenciamento em máquinas paralelas um dos fatores mais importantes a ser considerado é a restrição de recursos, no qual deve ser respeitada para se chegar ao melhor sequenciamento de tarefas. Segundo Pinedo (2012), o problema de sequenciamento em máquinas paralelas se relaciona diretamente à determinação da melhor sequência de execução dos Jobs, utilizando os recursos da melhor maneira possível.

Esse tipo de problema visa alocar diferentes tarefas em “n” máquinas, e tem grande relevância prática, pois muitas empresas utilizam em seu processo produtivo máquinas para a realização das tarefas. Portanto, a eficiência do processo de produção pode ser aprimorada por meio de estudos de sequenciamento em máquinas (COTA, 2018).

O problema em questão busca resolver 32 instâncias diferentes, com variados números de tarefas (Jobs) a serem alocados em “n” máquinas paralelas uniformes. Além disso, o problema possui uma restrição no qual cada instância possui uma determinada quantidade máxima de recursos que podem ser utilizados no processo.

Neste problema, assim como no problema apresentado por LOUREIRO (2014) às seguintes características podem ser definidas:

- As máquinas não possuem restrições referentes ao tipo de tarefa, ou seja, qualquer máquina pode executar qualquer tarefa;
- Cada máquina possui uma velocidade de produção e um custo por tempo diferente, portanto, o tempo de produção e o custo de uma tarefa depende da máquina que a executa;
- As máquinas só podem processar uma tarefa de cada vez;
- Não há prioridade entre as tarefas, elas possuem o mesmo grau de importância;
- Não há tempo de set-up;
- O tempo base de processamento de cada tarefa dado no início é constante, e não se altera conforme sua ordem de execução.

As máquinas possuem as características citadas anteriormente (velocidade e custo por tempo diferente) devido a não serem máquinas idênticas. E em uma situação real diversos elementos podem influenciar nisso, como a tecnologia empregada na máquina, as manutenções realizadas ao longo do tempo, entre outros fatores.

Outro ponto importante a ser ressaltado, cada tarefa possui um tempo base e a partir desse tempo base é feito o cálculo do tempo real, que é o tempo que a máquina leva para processar esse job.

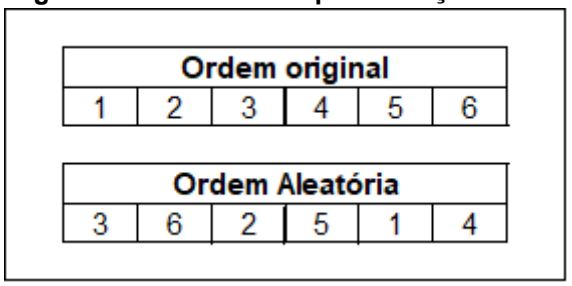
A função objetivo do problema busca minimizar o tempo necessário para a execução dos Jobs, ou seja, minimizar o Makespan (menor utilização de recursos) visando a menor quantidade de perdas, sempre respeitando a quantidade de recursos disponíveis. O Makespan é, de acordo com Baskar e Anthony (2014), o tempo total para a conclusão dos jobs e é considerado um dos principais indicadores de desempenho que necessita ser otimizado.

## 3. Proposta do Algoritmo

Neste trabalho, o algoritmo tem o objetivo de minimizar o Makespan (tempo total da operação) para o problema em questão. A solução será a indicação de quais jobs serão executados em cada máquina, respeitando sua restrição de recursos. O método *Hill-Climbing* foi usado para indicar a melhor solução para a instância. O algoritmo apresentado foi implementado na linguagem de programação Python.

Como o método *Hill-Climbing* parte do princípio de buscar a melhor solução dentre uma vizinhança, é necessário obter uma solução inicial aleatória. Para isso, primeiramente, foi encontrada a máquina mais eficiente, ou seja, aquela que possui o menor valor da divisão do custo tempo pela velocidade e a lista dos jobs é colocada em uma ordem aleatória (Figura 1). Em seguida, a primeira solução inicial é criada, em que todos os jobs, depois de aleatorizados, são inseridos na lista correspondente a máquina mais eficiente e as demais máquinas recebem listas vazias, conforme exemplo da Figura 2.

**Figura 1 - Lista de Jobs para solução inicial**

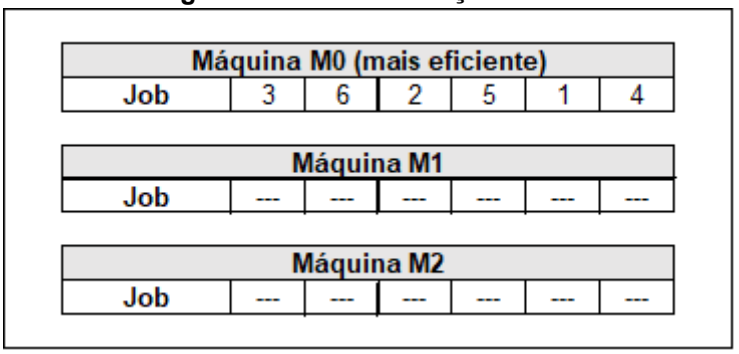


| Ordem original | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

| Ordem Aleatória | | | | | |
|---|---|---|---|---|---|
| 3 | 6 | 2 | 5 | 1 | 4 |

**Fonte: Os Autores (2021)**

**Figura 2 - Primeira solução inicial**



| Máquina M0 (mais eficiente) | | | | | | |
|---|---|---|---|---|---|---|
| Job | 3 | 6 | 2 | 5 | 1 | 4 |

| Máquina M1 | | | | | | |
|---|---|---|---|---|---|---|
| Job | --- | --- | --- | --- | --- | --- |

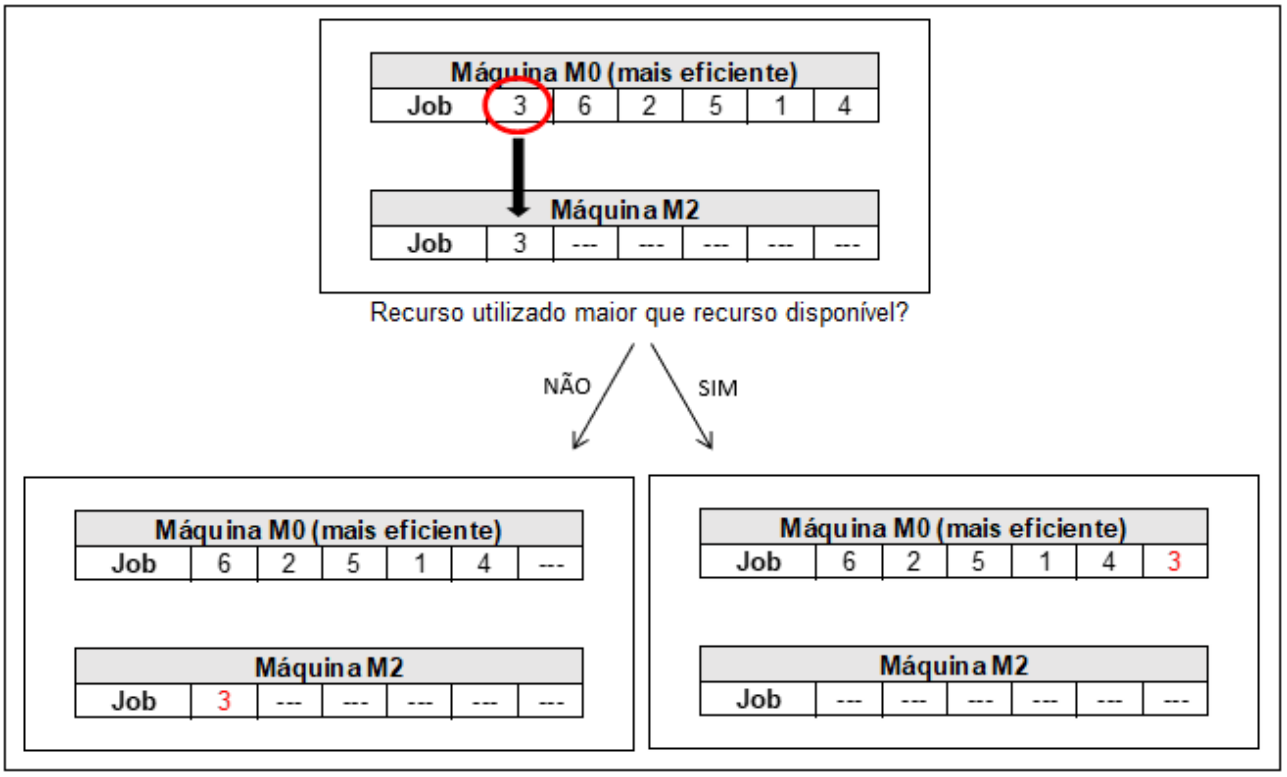
| Máquina M2 | | | | | | |
|---|---|---|---|---|---|---|
| Job | --- | --- | --- | --- | --- | --- |

**Fonte: Os Autores (2021)**

Porém, esta solução deve ser melhorada e tornada aleatória. Portanto, para cada job na máquina mais eficiente, na ordem em que foi alocado, é escolhida aleatoriamente uma outra máquina, sem ser a mais eficiente, a fim de tentar transferir o job de máquina. O primeiro job da solução inicial é tirado da máquina mais eficiente e colocado nela. Então, o recurso utilizado é calculado por meio da divisão do tempo base da máquina pela multiplicação da velocidade e do custo tempo do job em questão. Caso este recurso utilizado seja maior que o disponível, o job é tirado da máquina e devolvido no fim da lista de jobs da máquina mais eficiente. Após todas as iterações serem feitas, o recurso utilizado total é calculado e makespan da solução aleatória inicial são calculados.

O segundo passo do método *Hill-Climbing*, é o de geração de vizinhanças, que no algoritmo em questão se considerou a máquina com maior tempo de utilização e a máquina com menor tempo de utilização, tomando como base a solução aleatória. Para cada job da máquina que representa o makespan, é colocado o job na máquina com menor tempo consumido, a fim de reduzir o makespan. A partir disso, é calculado o makespan do vizinho gerado, se o recurso do vizinho gerado atender a restrição de recurso e o makespan for menor que o melhor vizinho gerado até o momento, o melhor vizinho é atualizado com a distribuição do novo vizinho gerado, e os valores de makespan e recurso utilizado são atualizados.

**Figura 3 - Exemplo de iteração para solução aleatória**




**Fonte: Os Autores (2021)**

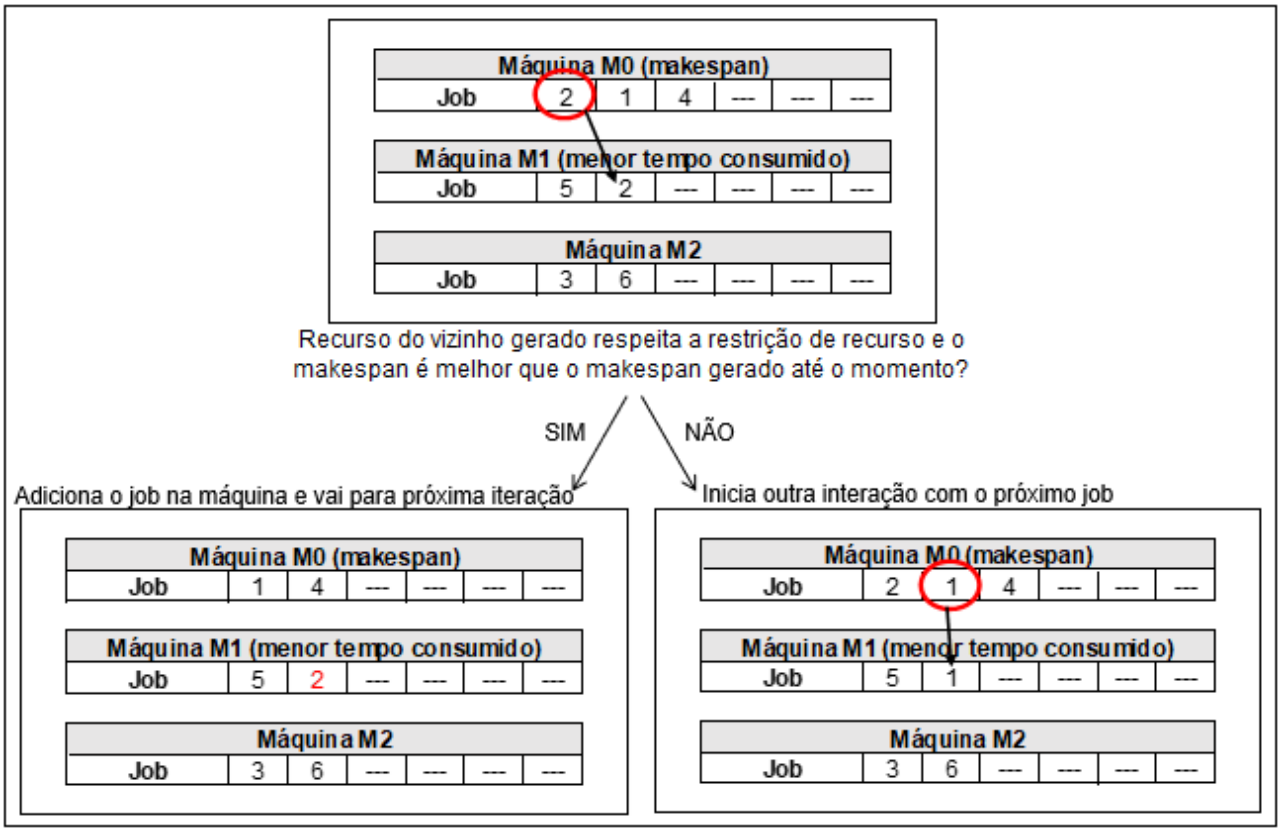
A Figura 4 representa a busca do melhor vizinho no *Hill-Climbing* em um problema de minimização, e a Figura 5 apresenta o método utilizado no algoritmo.

**Figura 4 - Busca do melhor vizinho**


**Fonte: Os Autores (2021)**

Importante ressaltar que na primeira iteração de geração de vizinhança, o makespan do melhor vizinho é o makespan da solução aleatória inicial, nas demais iterações o makespan vai corresponder ao makespan da melhor solução encontrada até o momento. Contudo, a máquina considerada mais ociosa e a com mais ocupação são fixas, com base na solução inicial.

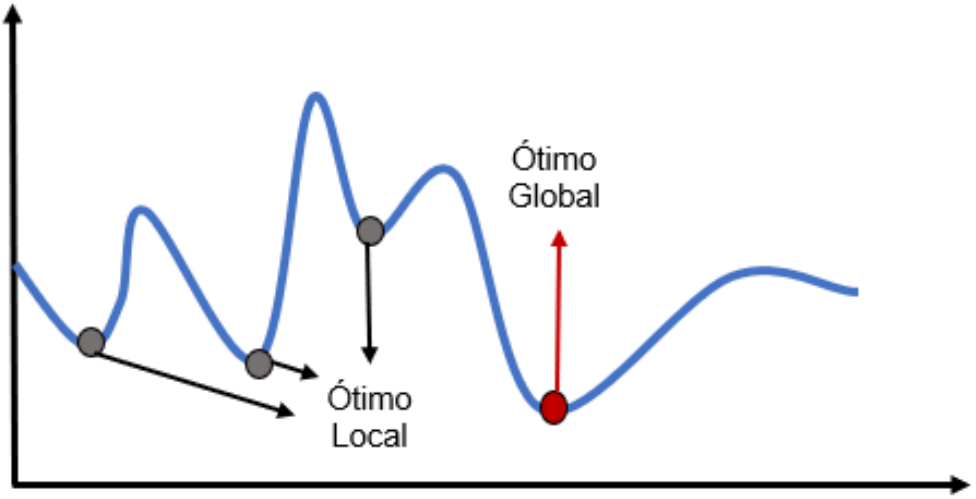
**Figura 5 - Iteração de busca do melhor vizinho**




**Fonte: Os Autores (2021)**

A partir dessa primeira geração de vizinhanças, são feitas outras iterações que fazem uma geração de vizinhos com base nos vizinhos encontrados na iteração anterior até que se chegue no ótimo local (Figura 6).

**Figura 6 - Esquematização ótimo local e ótimo global**




**Fonte: Os Autores (2021)**

Contudo, o que se deseja encontrar é o ótimo global, conforme exemplificado na Figura 6, para isso o algoritmo realizará um looping de acordo com a quantidade de iterações que for solicitada. Ou seja, se forem solicitadas 1000 iterações, como foi o caso estudado, o algoritmo fará 1000 vezes o looping, iniciando na geração de uma solução aleatória, passando pela geração de vizinhança, depois pela geração de vizinhos de vizinhos e terminando após realizar essas 1000 iterações, no qual o makespan ótimo será o menor valor encontrado dentre todas essas iterações.

## 4. Resultados Obtidos

O algoritmo descrito no item 3 deste artigo foi testado em 32 instâncias, geradas a partir do método de geração de instâncias descrito por Yeh *et al* (2015), nas quais variam a quantidade de máquinas, jobs e recursos disponíveis, o tempo base de cada job, a velocidade e o custo de cada máquina.

Todas elas foram rodadas em 1000 iterações no método *Hill-Climbing*, em busca do melhor resultado de makespan entre os vizinhos. Na tabela 1 estão apresentados o valor médio entre as 1000 iterações da melhora da solução local em relação à inicial.

**Tabela 1 – Média de melhora das soluções iniciais e locais das 1000 instâncias**

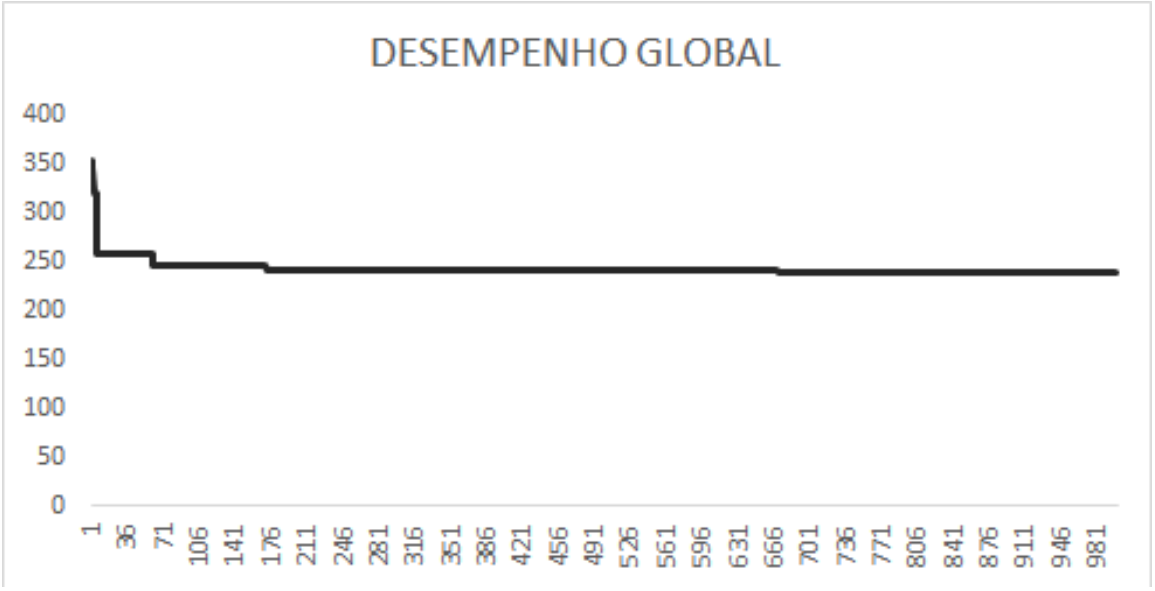
| Instância | Nº de máquinas | Nº de jobs | Melhora (%) |
|---|---|---|---|
| 1 | 3 | 50 | 21,94% |
| 2 | 3 | 50 | 17,95% |
| 3 | 3 | 50 | 22,98% |
| 4 | 3 | 50 | 0,37% |
| 5 | 3 | 100 | 16,17% |
| 6 | 3 | 100 | 29,17% |
| 7 | 3 | 100 | 0,30% |
| 8 | 3 | 100 | 15,12% |
| 9 | 5 | 100 | 0,03% |
| 10 | 5 | 100 | 41,58% |
| 11 | 5 | 100 | 37,07% |
| 12 | 5 | 100 | 0,01% |
| 13 | 5 | 200 | 0,00% |
| 14 | 5 | 200 | 0,01% |
| 15 | 5 | 200 | 0,02% |
| 16 | 5 | 200 | 35,79% |
| 17 | 5 | 300 | 31,46% |
| 18 | 5 | 300 | 0,00% |
| 19 | 5 | 300 | 11,12% |
| 20 | 5 | 300 | 25,50% |
| 21 | 10 | 100 | 10,07% |
| 22 | 10 | 100 | 17,50% |
| 23 | 10 | 100 | 0,01% |
| 24 | 10 | 100 | 18,41% |
| 25 | 10 | 200 | 20,12% |
| 26 | 10 | 200 | 23,92% |
| 27 | 10 | 200 | 0,01% |
| 28 | 10 | 200 | 6,11% |
| 29 | 10 | 300 | 17,11% |
| 30 | 10 | 300 | 0,00% |
| 31 | 10 | 300 | 19,22% |
| 32 | 10 | 300 | 7,54% |

**Fonte: Os autores (2021)**

Analisando a tabela, é possível perceber que algumas instâncias possuíram um maior valor de melhora que outras. Dessa forma, o valor médio de melhora entre elas foi de 13,96%.

Em cada uma das iterações, eram retornados valores da solução inicial, local e global. Com as soluções globais pode-se gerar um gráfico de linha que mostra o comportamento do *Hill-Climbing* na determinada instância. Para exemplificar, a figura 7 representa o comportamento ao longo das iterações da instância 10, a qual apresentava o maior índice de melhora.

**Figura 7 – Desempenho global das 1000 iterações na instância 10**




**Fonte: Os autores (2021)**

Na figura 7, observa-se o comportamento das soluções globais, que são os melhores valores atingidos até então, ao longo das iterações. Ela começa alta, possui uma grande queda nas primeiras iterações e depois mantém-se quase constante, com pequenas quedas conforme as novas iterações vão ocorrendo. Isso significa que, conforme novos vizinhos vão sendo gerados, novos valores de solução são encontrados e alguns deles conseguem superar o ótimo atingido até então.

Ainda analisando a solução inicial, pode-se fazer a comparação entre ela e a solução ótima obtida por meio da resolução do Solver Gurobi. Dessa forma, encontra-se a acurácia da solução inicial. De acordo com Yeh *et al* (2015), a modelagem para encontrar a solução ótima por meio do Solver Gurobi consiste em *n* jobs independentes $J = \{J_1,....,J_n\}$, prontos para serem processados em *m* máquinas uniformes paralelas $M = \{M_1,....,M_m\}$. O tempo de processamento de $J_j$ é $p_j$, a velocidade de $M_i$ é $s_i$, e o custo por unidade de tempo de $M_i$ é $\beta_i$. Dado um custo total que não pode ser excedido, o objetivo é encontrar uma alocação que minimize o Makespan. O problema pode ser formulado da seguinte maneira:

Min $C_{max}$,

sujeito a

$$\sum_{i=1}^{m} x_{ij} = 1 \; para \; j = 1,...., n,$$

$$\sum_{j=1}^{n} p_j x_{ij} / s_i \leq C_{max} \; para \; i = 1,...., m,$$

$$\sum_{j=1}^{n} \sum_{i=1}^{m} \beta_i p_j x_{ij} / s_i \leq B,$$

em que $x_{ij}$ é 1 se o job $J_j$ é designado para a máquina $M_i$, e 0 caso contrário.

Na tabela 2 está apresentada a média dos valores de solução inicial para as 1000 iterações de cada uma das instâncias, assim como a porcentagem da diferença entre ela e a Gurobi.

**Tabela 2 – Comparação entre solução inicial e solução ótima**

| Instância | Solução inicial | Solução Gurobi | Diferença (%) |
|---|---|---|---|
| 1 | 370,05 | 192,75 | 91,98% |
| 2 | 232,36 | 177,28 | 31,07% |
| 3 | 592,11 | 296,06 | 99,99% |
| 4 | 286,93 | 241,71 | 18,70% |
| 5 | 491,11 | 328,51 | 49,50% |
| 6 | 936,23 | 479,25 | 95,35% |
| 7 | 458,50 | 438,11 | 4,65% |
| 8 | 1154,3 | 526,05 | 119,4% |
| 9 | 494,68 | 322,36 | 53,45% |
| 10 | 622,50 | 233,16 | 166,9% |
| 11 | 574,91 | 260,60 | 120,6% |
| 12 | 953,73 | 847,77 | 12,50% |
| 13 | 1800,3 | 941,79 | 91,16% |
| 14 | 1592,5 | 970,53 | 64,10% |
| 15 | 1512,6 | 668,45 | 126,3% |
| 16 | 1171,2 | 480,56 | 143,7% |
| 17 | 1597,3 | 763,03 | 109,3% |
| 18 | 2553,6 | 2527,9 | 1,02% |
| 19 | 1448,19 | 939,61 | 54,13% |
| 20 | 1734,1 | 801,57 | 116,3% |
| 21 | 242,37 | 125,11 | 93,72% |
| 22 | 298,01 | 136,60 | 118,2% |
| 23 | 649,5 | 371,29 | 74,93% |
| 24 | 298,43 | 129,03 | 131,3% |
| 25 | 511,36 | 221,30 | 131,3% |
| 26 | 480,90 | 231,25 | 107,9% |
| 27 | 1808,17 | 1371,1 | 31,87% |
| 28 | 401,66 | 252,48 | 59,08% |
| 29 | 830,80 | 395,21 | 110,2% |
| 30 | 1623,4 | 555,85 | 192,1% |
| 31 | 760,63 | 352,07 | 116,0% |
| 32 | 655,95 | 406,44 | 61,39% |

**Fonte: Os autores (2021)**

A partir destes valores de diferença, chega-se a uma média de 87,44%, o que mostra que as soluções iniciais são pouco acuradas, fazendo o método *Hill-Climbing* ser necessário para encontrar uma boa solução para o problema.

A tabela 3 mostra a solução de cada uma das instâncias, obtidas com algoritmo apresentado, o resultado obtido pelo algoritmo Gurobi e a diferença entre eles (GAP). Esta diferença mostra o quão eficaz é o algoritmo apresentado neste artigo. Além disso, também estão apresentados o tempo (em segundos) que cada instância levou para chegar à solução, os recursos utilizados no algoritmo apresentado e na solução ótima.

Destes resultados, é possível observar que o algoritmo apresentado neste artigo teve um GAP médio de 21,78% e um tempo médio de processamento de 62,54 segundos. O tempo total para rodar todas as instâncias foi de 2063,76 segundos, ou 34,40 minutos. Sobre os recursos, é possível observar que a quantidade de recursos usados pode variar em relação à solução ótima e isso não significa que o makespan ficará ruim, e o contrário também é válido.

**Tabela 3 – Comparação entre a solução do algoritmo e a solução ótima**

| Instância | Solução algoritmo | Solução Gurobi | GAP | Tempo (s) | Recursos algoritmo | Recursos ótima |
|---|---|---|---|---|---|---|
| 1 | 192,87 | 192,75 | 0,06% | 7,54 | 1792,18 | 1792,37 |
| 2 | 177,30 | 177,28 | 0,01% | 6,22 | 1678,73 | 1678,71 |
| 3 | 296,19 | 296,06 | 0,04% | 8,58 | 2238,56 | 2237,99 |
| 4 | 242,06 | 241,71 | 0,14% | 3,05 | 1956,39 | 1957,00 |
| 5 | 328,70 | 328,51 | 0,06% | 36,91 | 2309,58 | 2309,36 |
| 6 | 480,85 | 479,25 | 0,33% | 50,77 | 6141,69 | 6143,70 |
| 7 | 438,14 | 438,11 | 0,01% | 10,35 | 1934,96 | 1935,00 |
| 8 | 661,01 | 526,05 | 25,66% | 37,71 | 5084,32 | 4407,69 |
| 9 | 387,96 | 322,36 | 20,35% | 11,78 | 1970,97 | 1970,99 |
| 10 | 239,62 | 233,16 | 2,77% | 23,88 | 3599,18 | 3637,00 |
| 11 | 277,27 | 260,60 | 6,40% | 21,29 | 3904,56 | 3952,89 |
| 12 | 875,80 | 847,77 | 3,31% | 20,04 | 3297,70 | 3297,97 |
| 13 | 1596,08 | 941,79 | 69,47% | 71,30 | 6003,97 | 6003,98 |
| 14 | 1464,99 | 970,53 | 50,95% | 55,31 | 6359,50 | 6359,95 |
| 15 | 1207,26 | 668,45 | 80,91% | 50,19 | 9402,90 | 9402,81 |
| 16 | 510,99 | 480,56 | 6,33% | 113,35 | 9942,21 | 9870,23 |
| 17 | 815,92 | 763,03 | 6,93% | 330,61 | 9686,84 | 9391,93 |
| 18 | 2529,54 | 2527,89 | 0,07% | 177,82 | 5884,48 | 5885,96 |
| 19 | 1072,59 | 939,61 | 14,15% | 168,46 | 16915,60 | 16329,34 |
| 20 | 940,38 | 801,57 | 17,32% | 284,93 | 10237,06 | 9497,90 |
| 21 | 140,57 | 125,11 | 12,36% | 7,32 | 3800,95 | 3859,98 |
| 22 | 176,60 | 136,60 | 29,28% | 7,73 | 3831,36 | 3722,98 |
| 23 | 555,70 | 371,29 | 49,67% | 17,43 | 1891,22 | 1897,95 |
| 24 | 152,29 | 129,03 | 18,03% | 7,52 | 4348,43 | 4275,40 |
| 25 | 248,97 | 221,30 | 28,77% | 29,83 | 5768,90 | 5381,50 |
| 26 | 268,36 | 231,25 | 16,00% | 31,94 | 4941,86 | 4772,58 |
| 27 | 1730,18 | 1371,12 | 26,19% | 82,17 | 3147,94 | 3147,98 |
| 28 | 275,94 | 252,48 | 9,29% | 21,67 | 5370,03 | 5387,50 |
| 29 | 506,30 | 395,21 | 28,15% | 71,64 | 12735,55 | 12048,81 |
| 30 | 1371,71 | 555,85 | 146,78% | 110,83 | 7210,52 | 7210,90 |
| 31 | 418,37 | 352,07 | 18,83% | 75,39 | 12327,01 | 11983,30 |
| 32 | 441,18 | 406,44 | 8,55% | 47,66 | 11652,30 | 11586,35 |

**Fonte: Os autores (2021)**

Quando analisados os valores de melhora da Tabela 1, de diferença na Tabela 2 e de GAP da Tabela 3, o comportamento esperado era a similaridade e coerência nos valores. No caso do valor médio de melhora ser próximo de zero, pode significar que a maioria das soluções iniciais eram próximas à ótima obtida pelo método ou que a geração de vizinhanças não foi eficaz para melhorar a solução. Porém, houve casos de baixa taxa média de melhora, nos quais o makespan final teve um GAP baixo, o que indica que a quantidade de iterações foi relevante para encontrar o valor melhor global próximo ao objetivo de referência. A acurácia da solução inicial também é essencial neste caso, porque, quando não há uma acurácia grande, o *Hill-Climbing* é o principal responsável pela melhora dela, então, se o algoritmo utilizado não foi muito eficiente para determinadas instâncias, essa solução inicial não será melhorada da forma com deve.

Entretanto, isso não indica a má qualidade do modelo, visto que o GAP médio ficou relativamente baixo, mostrando a similaridade média dos valores de solução encontrados com os considerados ótimos.

## 5. Conclusões

O problema de Sequenciamento em Máquinas Paralelas Uniformes com Restrição de Recursos é um dos inúmeros problemas que um Engenheiro de Produção pode encontrar na sua rotina em uma indústria. O entendimento deste problema não é complexo, porém sua implementação é, necessitando de alternativas para realizá-lo. A importância deste

modelo se dá por conta de ser comum as indústrias trabalharem com várias máquinas para realização de diferentes tarefas, buscando otimizar o processo e economizar tempo e recursos.

Nesse sentido, este artigo teve como objetivo apresentar um algoritmo capaz de encontrar uma boa solução a este tipo de problema, facilitando o trabalho de quem precisar lidar com ele. O trabalho foi implementado em linguagem de programação Python.

Mesmo com o bom desempenho do algoritmo *Hill-Climbing*, os resultados de makespan encontram-se, em média, com 21,78% de GAP em relação às soluções ótimas, e demoraram um tempo significativo para encontrá-las, com 1000 iterações, o que mostra que, mesmo que eficiente, ainda é possível criar outros modelos que encontram soluções mais próximas às ótimas, a partir de uma distribuição diferente de jobs na construção da solução inicial ou em uma estrutura diferente de vizinhança, em que a máquina considerada mais ociosa e a com mais ocupação sejam atualizadas a cada melhor vizinho e não sejam fixas com base na solução inicial, desse modo, essas hipóteses podem ser testadas em estudos futuros.

## REFERÊNCIAS


A., Baskar, M., Anthony Xavior. Optimization of Makespan in Job and Machine Priority Environment. **Procedia Engineering**. 97 (2014), 22-28.

ABREU, J. C.; PEREIRA, A. A. S. Meta-heurística multiobjetivo para sequenciamento de máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência. **Revista científica FAGOC Multidisciplinar.** v.3, P. 31-43

BENNETT, A. P. When a genetic algorithm outperforms hill-climbing. **Theoretical Computer Science**, 320, p. 135-153, 2004.

COLNAGO, A. C.,de MORAIS, A. P. B., CARDOSO, M. S., BOTELHO, G. O., LIMA, R. H. P. Desenvolvimento de um algoritmo Hill-Climbing para minimização do makespan em Problemas de Sequenciamento em FlowShops. In: IX Congresso Brasileiro de Engenharia de Produção. **Anais do IX ConBRepro.** Ponta Grossa-PR, 2019.

COTA, L. P. **Abordagens exatas e heurísticas para o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência**. Belo Horizonte, 162 p., 2018. Tese (Doutorado) - Universidade Federal de Minas Gerais.

LI, K.; CHEN, J.; FU, H.; JIA, Z.; FU, W. Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. **Applied Soft Computing Journal,** 82 (2019).

LOUREIRO, N. F. P. **Utilização do Simulated Annealing na resolução de problemas no planejamento de produção**. Coimbra, 62 p., 2014. Dissertação (Mestrado) - Universidade de Coimbra.

Mundim, L. R; Fuchigami, H. Y. Uma heurística robusta para programação de máquinas paralelas com tempos de setup dependentes da sequência. **Revista Produção Online**, v.17, n.2, p. 463-481, 2017.

PINEDO, M. L. **Scheduling: theory, algorithms, and systems**. Boston, MA: Springer US, 2012. 694 p.

YEH, W. C.; CHUANG, M. C.; LEE, W. C. Uniform parallel machine scheduling with resource consumption constraint. **Applied Mathematical Modelling**. 39 (2015) P. 2131-2138.