XXVIII Congresso {virtual} de Iniciação Científica da Unicamp 2020

Algoritmos para o Problema de Job-Shop

Aluno: Wilson Yang
w207274@dac.unicamp.br

Orientadora: Profa. Dra. Priscila Cristina Berbert Rampazzo
Faculdade de Ciências Aplicadas

RESUMO

Diversos problemas operacionais de indústrias de produtos e serviços são causados pela alocação indevida de recursos. A área de otimização apresenta os problemas de alocação e sequenciamento de tarefas a um conjunto de máquinas disponíveis através da classe de *Scheduling Problems*. Dentro das diversas classificações que esta classe representa, o intuito deste projeto é pesquisar, propor e implementar soluções para o problema de *Job Shop*. Na primeira etapa do projeto, o problema foi modelado matematicamente de diferentes formas e resolvido por dois *solvers* de Programação Matemática: CBC e CPLEX. Concomitantemente, foi implementada uma heurística simples, a Heurística do *Bottleneck* como alternativa para a obtenção de uma solução para o problema de *Job Shop*, em linguagem de Programação Python. Na segunda etapa deste projeto, foi proposta uma metaheurística populacional: um Algoritmo Genético. Além de serem mais elaborados que heurísticas simples, os Algoritmos Genéticos trabalham com um conjunto de soluções, simultaneamente, permitindo a obtenção de soluções alternativas em uma única execução do algoritmo. Por fim, todos os resultados obtidos serão analisados para comparação entre a eficácia de cada método.

PALAVRAS CHAVE. ***Scheduling*, Tarefas, Máquinas, *Job Shop*, Algoritmos, Indústria 4.0.**

1. Introdução

O problema clássico de *Job Shop* [Pinedo, 2010; Brucker, 2006] consiste de um conjunto de máquinas diferentes que devem processar operações de um conjunto de tarefas. Cada tarefa possui um fluxo específico de operações, uma rota de máquinas própria e fixa, previamente conhecida. As operações das tarefas não podem ser interrompidas e devem respeitar a ordem de precedência da rota. Além disso, as restrições de alocação devem ser satisfeitas: uma operação só pode ser processada por uma máquina e cada máquina pode processar apenas uma operação por vez. Dentre as possíveis funções-objetivo do problema [Leung, 2004], podemos destacar: a minimização do *makespan*, a minimização do tempo de espera, a minimização do tempo de atraso total, a minimização do atraso máximo, etc. De acordo com a formulação adotada (fundamentalmente dependente das restrições, parâmetros e quantidade de máquinas) podemos resolver o problema de forma exata em tempo polinomial, ou ainda lidar com problemas da classe NP, sem conhecimento de uma solução exata para o problema. Nestes casos, é necessário abandonar a busca de uma solução ótima e simplesmente procurar uma solução de qualidade, através de procedimentos heurísticos.

2. Métodos e Metodologia

Considere o problema com n tarefas e m máquinas. A tarefa i passa por n_i operações com relação de precedência: $O_{i1}, O_{i2}, ..., O_{i,n_i}$, $(h = 1, 2, ...n_i)$. Existe uma máquina $j = \mu_{ih} \in \{M_1, ..., M_m\}$ com tempo de processamento p_{ij} associada a cada operação O_{ih}. A operação O_{ih} deve ser processado em p_{ij} unidades de tempo na máquina $j = \mu_{ih}$. Define-se C_i como o tempo de término da última operação O_{i,n_i} [Brucker, 2006; Pinedo, 2010].

Esse problema problema é resolvido com dois modelos matemáticos, uma heurística e uma metaheurística.

2.1. Modelo matemático 1

As variáveis de decisão de um modelo clássico podem ser definidas como:

$$x_{ij} = \text{início da tarefa } i \text{ na máquina } j.$$

$$w_{ikj} = \begin{cases} 1 & \text{se tarefa } i \text{ precede tarefa } k \text{ na máquina } j. \\ 0 & \text{caso contrário.} \end{cases}$$

Assim, o seguinte problema que minimiza o makespan:

$$\min \quad Cmax \tag{1}$$

$$\text{s.a} \quad Cmax \geqslant x_{ij} + p_{ij} \quad \forall i = 1, ..., n \quad \forall h = 1, ..., n_i \quad j = \mu_{ih} \tag{2}$$

$$x_{ij} \geqslant x_{il} + p_{il} \quad \forall i = 1, ..., n \quad \forall h = 2, ..., n_i \quad j = \mu_{ih} \quad l = \mu_{i(h-1)} \tag{3}$$

$$x_{ij} + p_{ij} \leqslant x_{kj} + M(1 - w_{ikj}) \quad \forall i = 1, ..., n \quad \forall k > i \quad \forall j \in \{\mu_{ih_i}\} \cap \{\mu_{kh_k}\} \quad \forall h_i = 1, ..., n_i \quad \forall h_k = 1, ..., n_k \tag{4}$$

$$x_{kj} + p_{kj} \leqslant x_{ij} + Mw_{ikj} \quad \forall i = 1, ..., n \quad \forall k > i \quad \forall j \in \{\mu_{ih_i}\} \cap \{\mu_{kh_k}\} \quad \forall h_i = 1, ..., n_i \quad \forall h_k = 1, ..., n_k \tag{5}$$

$$x_{ij} \geqslant 0 \quad \forall i = 1, ..., n \quad \forall j = 1, ..., m$$

$$w_{ikj} \in \{0, 1\} \quad \forall i = 1, ..., n \quad \forall k > i \quad \forall j = 1, ..., m$$

2.2. Modelo matemático 2

Alternativamente, outras modelagens serão exploradas e implementadas. O problema de *Job Shop* poderia ser representado através de um grafo. Nesta representação, todas as operações de uma mesma tarefa são conectadas por arcos conjuntivos. Arcos disjuntivos ligam operações a serem processadas numa mesma máquina. A decisão básica desta alocação é ordenar as operações em cada máquina, o que equivale a tornar os arcos bidirecionados (disjuntivos) em direcionados.

Considerando esta representação, outro modelo pode ser definido:

$$x_{ij} = \text{tempo de início da operação } O_{ij}.$$

$$w_d = \begin{cases} 1 & \text{se a operação } O_{ij} \text{ acontece antes da operação } O_{kj}. \\ 0 & \text{caso contrário.} \end{cases}$$

N: conjunto de todas as operações.
A: conjunto das restrições conjuntivas.
D: conjunto das restrições disjuntivas.
$(i, j) \rightarrow (i, l) \in A$ (arcos conjuntivos).
$d = (i, j) \leftrightarrow (k, j) \in D$ (arcos disjuntivos).

$$\min \quad Cmax \tag{6}$$

$$\text{s.a} \quad Cmax \geqslant x_{ij} + p_{ij} \quad \forall (i,j) \in N \tag{7}$$

$$x_{il} \geqslant x_{ij} + p_{ij} \quad \forall (i,j) \rightarrow (i,l) \in A \tag{8}$$

$$x_{ij} + p_{ij} \leqslant x_{kj} + M(1 - w_d) \ \forall d = (i,j) \leftrightarrow (k,j) \in D \tag{9}$$

$$x_{kj} + p_{kj} \leqslant x_{ij} + M w_d \ \forall d = (i,j) \leftrightarrow (k,j) \in D \tag{10}$$

$$x_{ij} \geqslant 0 \quad \forall (i,j) \in N$$

$$w_d \in \{0,1\} \quad \forall d = (i,j) \leftrightarrow (k,j) \in D$$

2.3. Heurística do *Bottleneck*

Um dos procedimentos heurísticos mais bem-sucedidos desenvolvidos para o problema de Job Shop é a Heurística do *Bottleneck* [Pinedo, 2010]. Em uma determinada iteração da heurística, assume-se que nas iterações anteriores um conjunto de arcos disjuntivos já foi fixado para um subconjunto M_0 de máquinas. Portanto, para cada máquina em M_0, uma sequência de operações já foi determinada. Depois de resolver vários problemas de máquina única, a máquina com o maior atraso máximo é escolhida. Entre as máquinas restantes, essa máquina é a mais crítica ou o "gargalo" e, portanto, a próxima a ser incluída em M_0. O makespan pode ser calculado através da resolução de um problema de Caminho Crítico [Taha, 2007] para o grafo, após a inserção de todas as máquinas em M_0.

2.4. Algoritmo Genético

O Algoritmo Genético (GA) é uma metaheurística populacional baseada em mecanismos da evolução biológica [Bäck et al., 2000]. O GA, inspirado na teoria da seleção natural, trabalha com um conjunto finito inicial de soluções factíveis do problema de *Job Shop*, chamado de população inicial. Ao longo das gerações, a população se aprimora no sentido de obter melhores valores da função-objetivo, devido à pressão seletiva. O esquema geral do GA está representado na Figura 1. Na codificação, para cada tarefa i, o vetor indivíduo será preenchido com o número i, n_i vezes, aleatoriamente. Ao decodificar o indivíduo, será conhecida a sequência de processamento, pois, ao aparecer o número i pela primeira vez, isso representa a operação $O_{i,1}$, na segunda vez, seria a operação $O_{i,2}$ e assim por diante. Foram implementados e testados: 2 formas de inicialização, 3 tipos de crossover, 5 tipos de mutação e seleção por torneio binário. Os parâmetros e configurações do algoritmo são definidos de forma empírica: tipos de inicialização, *crossover* e mutação, de forma a buscar a melhor versão para o GA, ou seja, a que apresentou a melhor função-objetivo em cada instância.

3. Apresentação dos Resultados

Todos os programas foram implementados na linguagem de programação Python. Após a modelagem, podemos usar os *solvers* comerciais, como o CPLEX[1], ou os de código aberto, como o CBC[2] para resolver o problema. Os *solvers* tiveram tempo de execução limitado em 45 minutos. As instâncias de testes escolhidas estão disponíveis em um repositório on-line[3]: *Demirkol's instances for Job Shop Scheduling Problem*. Vale destacar que o tempo médio de uma rodada do Algoritmo

[1] IBM ILOG CPLEX Optimization Studio, versão acadêmica.

[2] utilizado com a importação do pacote *ortools*. O OR-Tools [OR-Tools, 2019] é uma biblioteca de código aberto para escrever os modelos de otimização.

[3] `http://optimizizer.com/DMU.php`

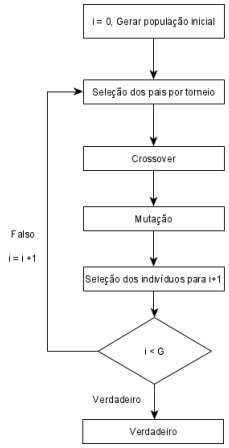

Figura 1: Esquema geral do GA

Genético, para a maior instância foi de aproximadamente 5,6 minutos para a obtenção de uma solução de qualidade.

As Tabelas a seguir, resumem os resultados obtidos[4] até o momento, com os seguintes modos de resolução:

- (M1-CBC) Modelo 1 + *solver* CBC
- (M2-CBC) Modelo 2 + *solver* CBC
- (M1-CPLEX) Modelo 1 + *solver* CPLEX
- (M2-CPLEX) Modelo 2 + *solver* CPLEX
- (HBk) Heurística do *Bottleneck* - CBC
- (HBk) Heurística do *Bottleneck* - CPLEX
- (GA) Algoritmo Genético proposto.

Analisando os resultados de todos os modos de resolução, indicamos a seguir, a abordagem que apresentou os melhores resultados:

- instância 1, $n = 10$, $m = 15$, *makespan* = 2113 (M1-CPLEX e M2 -CPLEX).
- instância 2, $n = 20$, $m = 15$, *makespan* = 2736 (M1-CPLEX)
- instância 3, $n = 30$, $m = 15$, *makespan* = 4021,14 (GA)
- instância 4, $n = 40$, $m = 15$, *makespan* = 5059,43 (GA)
- instância 5, $n = 50$, $m = 15$, *makespan* = 6102,57 (GA)
- instância 6, $n = 50$, $m = 20$, *makespan* = 6876,29 (GA)

[4]Os testes foram executados em sistema operacional Linux, Intel(R) Core(TM) Intel Core i7-9700K CPU 3.60GHz, 16Gb de memória RAM.

O Algoritmo Genético proposto se destacou em relação ao tempo computacional, consideravelmente menor e na qualidade da solução factível apresentada em instâncias de grande porte, quando comparado aos modelos M1 e M2 resolvidos com CPLEX com limitação de tempo.

A Figura 2 apresenta o **Gráfico de Gantt**, uma ferramenta de controle e representação da produção. O gráfico foi produzido com as funções do módulo *plotly* em *Python*. Para melhor visualização, a representação refere-se à solução ótima do problema com 15 máquinas e 10 tarefas, com *makespan* igual a **2113**.

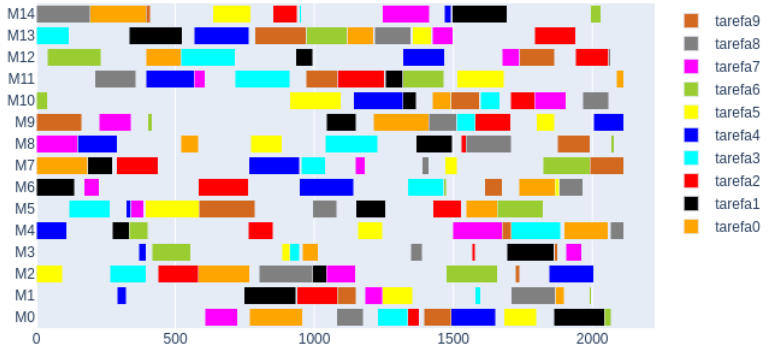

Figura 2: Gráfico de Gantt: Representação da solução ótima para a instância 1.

4. Conclusão

O problema do *Job Shop* pertence à classe de problemas NP-difícil e sua resolução de forma exata, por meio de *solvers* de Programação Matemática, é uma tarefa com alta complexidade computacional e que demanda muito tempo para a execução. Por isso, torna-se necessário abandonar a busca de uma solução ótima e procurar soluções de qualidade por meio de heurísticas e metaheurísticas.

A Heurística do *Bottleneck*, apesar de ser um dos procedimentos heurísticos mais conhecidos desenvolvidos para o problema do *Job Shop*, não se mostrou eficiente na obtenção de uma solução factível no tempo pré-definido (45 minutos). O Algoritmo Genético, por outro lado, conseguiu encontrar soluções alternativas com melhor *makespan* do que as soluções propostas pelos *solvers* e pela Heurística de *Bottleneck*, em um curto período de tempo (menos de 6 minutos).

Referências

Bäck, T., Fogel, D. B., e Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, UK, 1 edition.

Brucker, P. (2006). *Scheduling Algorithms*. Springer-Verlag B. H., New York, 5 edition.

Leung, J. Y.-T. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, United States, 4ªed. edition.

OR-Tools, G. (2019). Route. schedule. plan. assign. pack. solve. or-tools is fast and portable software for combinatorial optimization. https://developers.google.com/optimization/.

Pinedo, M. L. (2010). *Scheduling: Theory, Algorithms and Systems*. Springer, New York, 4 edition.

Taha, H. A. (2007). *Pesquisa Operacional*. Pearson (Prentice Hall), São Paulo, 8ªed. edition.