Estudo de um modelo de sequenciamento da produção

Aluna: Letícia Satie Morimoto (satie.leticia@gmail.com)
Orientador: Francisco A. M. Gomes (chico@ime.unicamp.br)
IMECC- Instituto de Matemática, Estatística e Computação Científica

Introdução

Neste projeto, formulamos e resolvemos problemas de seqüenciamento de produção. Nossa abordagem foi feita tanto do ponto de vista teórico como prático.

O nosso objetivo é a solução de problemas práticos da área de produção através das técnicas de modelagem e otimização vistas no curso de matemática aplicada da UNICAMP.

Formulamos e resolvemos problemas utilizando uma rotina de solução de problemas de programação inteira mista, denominada IP, além de uma metaheurística, o Variable Neighborhood Search (VNS), ambas implementadas no MATLAB. Finalmente, estudamos a combinação dos algoritmos VNS e IP, com finalidade de obtermos soluções melhores em menos tempo de processamento dos programas.

Figura 1: Motivação para o projeto.

Metodologia

Nosso problema consiste em seqüenciar nt tarefas sendo que as tarefas devem passar por todas as nm máquinas em ordem, ou seja, a máquina m não pode processar a tarefa i antes da máquina m-1 terminar de processar esta tarefa, além disso em cada máquina cada tarefa possui um tempo de processamento diferente. A formulação deste problema é dada a seguir:

Variáveis:

$$x_{i,j} = \begin{cases} 1, se\ tarefa\ i\ está\ na\ posição\ j \\ 0, caso\ contrário \end{cases}$$

$$t_{m,j}\ é\ o\ tempo\ de\ início\ de\ processamento\ da\ j\ -ésima\ tarefa$$

onde i=1, ..., nt; j=1, ..., nt; m=1, ...nm.

Função objetivo: $\min \quad t_{nm,nt} + \sum_{i=0}^{nt} p_{nm,i} x_{i,nt}$

Restrições:

Em cada posição só pode haver uma tarefa: $\sum_{i=0}^{nt} x_{i,j} = 1, \qquad \forall j = 1, \dots, nt$

Cada tarefa só pode estar em uma posição: $\sum_{j=0}^{nt} x_{i,j} = 1, \qquad \forall i = 1, \dots, nt$

A tarefa i só pode começar a ser processada em uma máquina depois que a tarefa i-1 terminar de ser processada nessa mesma máquina:

$$t_{m,j} \geq t_{m,j-1} + \sum_{i=0}^{nt} p_{m,i} x_{i,j-1} \quad \forall m = 1, \dots, nm\ e\ j = 2, \dots, nt$$

Uma tarefa só pode começar a ser processada na máquina m depois que terminar de ser processada na máquina m-1:

$$t_{m,j} \geq t_{m-1,j} + \sum_{i=0}^{nt} p_{m,i-1} x_{i,j} \quad \forall m = 2, \dots, nm\ e\ j = 1, \dots, nt$$

Tempo inicial: $t_{1,1} = 0$

Todos os instantes são positivos: $t_{m,j} \geq 0 \ \forall\, m = 1, \dots, nm$

O modelo adotado envolve a solução de problemas de programação inteira mista. Dada a complexidade desse tipo de problema e o fato dele envolver um grande número de variáveis, combinamos o algoritmo branch-and-bound fornecido pelo MATLAB com uma meta-heurística, a busca em vizinhança variável (VNS - Variable Neighborhood Search), cujo propósito é facilitar a obtenção de boas soluções inteiras, acelerando o processo de exclusão de ramos da árvore de busca do algoritmo.

A VNS é uma metaheurística baseada em uma mudança sistemática de vizinhanças combinada com uma busca local que procura encontrar o mínimo global de um problema.

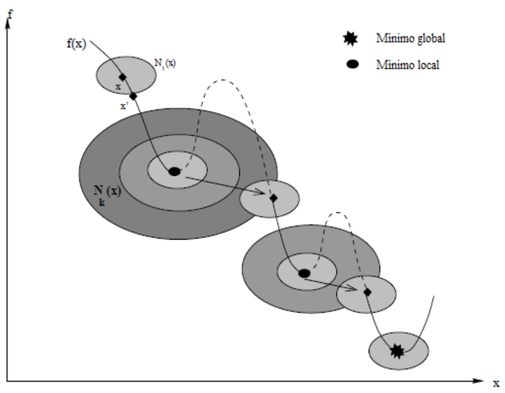

Figura 2: VNS básico.

Como podem existir muitos mínimos locais, com valores bastante diferentes, partimos de um mínimo local de uma vizinhança para encontrar o mínimo local em outra vizinhança. A idéia básica da VNS pode ser resumida em: Efetuar uma sequência de buscas locais, trocando a estrutura de vizinhança durante a busca, e explorando vizinhanças gradativamente maiores.

A estrutura das vizinhanças utilizadas é um dos elementos mais importantes na metaheurística, pois elas determinam sua eficiência. Para o problema de sequenciamento adotamos as seguintes estruturas de vizinhança:

Exchange: um vizinho é gerado escolhendo aleatoriamente duas tarefas e trocando-as de lugar.

Insert: um vizinho é gerado escolhendo aleatoriamente uma tarefa e uma posição. A tarefa escolhida é retirada de sua posição original e inserida na nova posição.

No algoritmo que implementamos, geramos uma solução inicial e, a partir dela, procuramos uma solução melhor em sua vizinhança. Quando uma das duas estruturas de vizinhança é esgotada, passamos à outra. O processo é repetido por k_{max} iterações, onde k_{max} é definido pelo usuário. Em nossos testes, usamos k_{max} = 1000.

O algoritmo foi implementado no pacote MATLAB. Ele recebe como entrada a matriz P (que armazena os tempos de processamento das tarefas nas máquina), além do número máximo de iterações e, opcionalmente, uma sequência inicial. Caso a sequencia inicial não seja fornecida, uma solução aleatória é gerada.

Para acelerar a resolução do problema pelo modelo matemático, iremos utilizar o VNS para encontrar uma solução inicial e utilizá-la como limitante superior.

Fizemos isso de duas formas:
colocando a rotina IP para chamar o VNS apenas uma vez no início (chamamos a rotina programada no MATLAB de IPVNS);
colocando a rotina IP para chamar o VNS a cada iteração do IP (chamamos a rotina programada no MATLAB de IPVNS2).

Como um efeito colateral dessa abordagem, também conseguimos analisar a qualidade da solução gerada pela VNS.

Resultados

Fizemos testes com problemas de diversos tamanhos a fim de verificar a eficiência de cada um dos métodos apresentados. As matrizes de tempo de processamento foram retiradas de um sítio de testes de *scheduling*.

Para um problema com 8 tarefas e 5 máquinas:

Método	Tempo (min) *	Mksp inicial	Melhor mksp
VNS puro	0,01	---	707
IP puro	----	---	---
IPVNS	1,92	706	704
IPVNS2	2,70	706	704

Para um problema com 10 tarefas e 5 máquinas:

Método	Tempo (min)	Mksp inicial	Melhor mksp
VNS puro	0,01	---	763
IP puro	7,34	---	763
IPVNS	0,78	768	763
IPVNS2	0,69	768	763

Nota-se que o VNS puro encontra uma solução boa em tempo muito menor que os outros métodos, no entanto, essa solução pode não ser ótima. Então, utilizando o IPVNS, que por sua vez utiliza o VNS para encontrar uma solução inicial, esse método demora, muitas vezes, o dobro do tempo para verificar se a solução é ótima, e para corrigi-la caso não seja. Enquanto o IPVNS2, gasta cerca de 5% a mais de tempo para encontrar a solução ótima utilizando o VNS para encontrar um limitante superior em cada iteração.
Já o IP puro, que resolve o problema de programação linear inteira mista não conseguiu encontrar solução pra problemas com mais de 7 tarefas e 5 máquinas, pois a quantidade de soluções viáveis cresce exponencialmente, tornando o problema extremamente difícil.

Conclusão

O uso de um algoritmo de programação inteira puro pode não ser eficiente para a resolução de problemas de sequenciamento, particularmente quando os problemas são de grande porte. Por outro lado, se combinamos a programação inteira com uma meta-heurística, o algoritmo fica mais eficiente, permitindo a resolução de problemas bem maiores.

Além disso, se uma solução boa, porém não ótima, já for suficiente em determinada ocasião, pode-se utilizar apenas a metaheurística, pois ela geralmente encontra soluções bastante próximas do ótimo em tempo consideravelmente menor.

Referências Bibliográficas

1. PINEDO, M. – *Scheduling: theory, algorithms and systems*. Englewood Cliffs, Prentice-Hall, 1995.
2. http://www.ise.ncsu.edu/kay/matlog/.
3. P. Hansen, N. Mladenovi´c,, J.A. Moreno P´erez, Variable Neighborhood Search: Methods and Applications, 2008
4. http://mistic.heigvd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html