Teoria da Decisão
Métodos Escalares de Otimização Vetorial e Tomada de Decisão Assistida

Rafael Carneiro de Castro
Engenharia de Sistemas - UFMG
Matrícula: 2013030210
Email: rafaelcarneiroget@hotmail.com

Davi Pinheiro Viana
Engenharia de Sistemas - UFMG
Matrícula: 2013029912
Email: daviviana22@gmail.com

***Resumo*—Abordagem de forma conjunta de grande parte dos conceitos vistos na disciplina "ELE088 - Teoria da Decisão", através de um problema de escalonamento de tarefas.**

I. Introdução

O presente trabalho tem o objetivo de resolver um problema de otimização utilizando as técnicas escalares e de decisão assistida estudados em sala de aula, colocando em prática grande parte dos conceitos da matéria.

O problema a ser resolvido é o seguinte: *Uma empresa possui um conjunto de M máquinas que devem ser utilizadas para processar N tarefas indivisíveis. Cada máquina i leva um tempo t_{ij} para processar uma tarefa j e pode processar uma única tarefa por vez. Todas as tarefas possuem uma mesma data ideal de entrega d, sendo que cada tarefa j sofre uma penalidade w_j proporcional a cada dia que ela é entregue adiantada ou atrasada em relação a d.*

A. Formulação do Problema:

A formulação do problema foi dividida em duas partes, como é discutido a seguir:

1) Minimização do Tempo Total de Entrega: Em primeiro momento é preciso construir uma função objetivo e suas eventuais restrições para minimização do tempo total de entrega de todas as tarefas. Considere C_i como sendo o tempo necessário para se terminar as tarefas executadas pela máquina i. Assim:

$$C_i = \sum_{j=1}^{N} t_{ij} * x_{ij} \ \forall \ i \in \ (1, ..., M)$$

O objetivo então se torna:

$$min \ C_{max}$$

$$C_{max} = max(C_i) \ \forall \ i \in \ (1, ..., M)$$

sujeito a:

$$\sum_{i=1}^{M} x_{ij} = 1 \ \forall \ j \in \ (1, ..., N)$$

$$\sum_{j=1}^{N} t_{ij} * x_{ij} <= C_{max} \ \forall \ i \in \ (1, ..., M)$$

$$x_{ij} \in (0, 1)$$

Com estas restrições, garante-se que cada tarefa vai ser cumprida por uma única máquina. A matriz x é composta por zeros e uns. Cada uma das suas linhas, então, vai representar uma tarefa, e cada coluna, uma máquina. O número 1 em uma linha representa qual máquina vai executar a tarefa daquela linha.

2) Minimização da Soma Ponderada dos Atrasos e Adiantamentos: Agora, uma função objetivo para tratar a minimização da soma ponderada dos atrasos e adiantamentos é formulada. O momento de término da tarefa j será chamado de e_j. Então:

$$e_j = \sum_{i=1}^{M} t_{ik} \ \forall \ k \in \ \Omega_i$$

onde Ω_i é o conjunto das tarefas até a tarefa i. A função objetivo pode ser escrita como:

$$min \ \sum_{j=1}^{N} w_j |e_j - d|$$

sujeito a:

$$\sum_{i=1}^{M} x_{ij} = 1 \ \forall \ j \in \ (1, ..., N)$$

$$x_{ij} \in (0, 1)$$

onde, como já discutido, d é a data ideal de entrega das tarefas e w_j a penalidade proporcional a cada dia que a tarefa é entregue adiantada ou atrasada em relação a d.

B. Algoritmos de Solução:

Nesta seção serão discutidos e exibidos os algoritmos para solução dos problemas mono e multi objetivo.

1) Minimização do Tempo Total de Entrega: O algoritmo de otimização utilizado aqui se baseia no *Simulated Annealing*, método estudado em sala de aula de fácil implementação e convergência atrativa. Este método escapa de mínimos locais com a aceitação de alguns movimentos de piora na qualidade da solução. É inspirado no recozimento físico de sólidos, e possui um parâmetro conhecido como *temperatura*, que ajusta a probabilidade de um movimento de piora ser aceito. Um algoritmo simplificado para o *Simulated Annealing* pode ser visto na Figura 1.

Algoritmo 1: Simulated Annealing

Defina um contator $k = 0$;
Defina uma temperatura inicial $t_k \geq 0$;
Defina T_k (função que controla a variação da temperatura);
Defina M_k (no. de iterações executadas na temperatura t_k);
Selecione uma solução inicial $\boldsymbol{x} \in \Omega$;
while *critério de parada não alcançado* **do**
    Defina o contator $m = 0$;
    while $m \leq M_k$ **do**
        Gere uma solução $\boldsymbol{x}' \in \mathcal{N}(\boldsymbol{x})$;
        Calcule $\Delta E = f(\boldsymbol{x}') - f(\boldsymbol{x})$;
        if $\Delta E \leq 0$ **then**
            $\boldsymbol{x} \leftarrow \boldsymbol{x}'$;
        else
            $\boldsymbol{x} \leftarrow \boldsymbol{x}'$ com probabilidade $exp(-\Delta E/t_k)$;
        $m \leftarrow m + 1$;
    $t_{k+1} \leftarrow T_k(t_k)$;
    $k \leftarrow k + 1$;

Figura 1. Algoritmo simplificado do SA.

Para tratar o problema da minimização do tempo de entrega, é importante ter em mente a representação de uma possível solução. Esta representação, como discutido na seção A.1, é uma matriz de zeros e uns, onde o 1 representa qual máquina faz dada tarefa.

A primeira etapa foi criar um algoritmo que gera uma solução inicial. Este código está no arquivo `initialSolTE.m`. Uma solução é inicializada como sendo uma matriz de zeros. Então, para cada linha (tarefa), um número randômico entre 1 e a quantidade de máquinas é gerado, representado qual é a máquina escolhida para executar a tarefa daquela linha. Um número 1 é colocado na posição da linha do número randômico gerado. Repare que esta solução gerada nunca viola a restrição de que a soma dos valores de uma linha deve ser sempre 1.

Em seguida, criou-se um código que é responsável por avaliar uma dada solução na função objetivo, algoritmo este que está no arquivo `fobjTE.m`. Este arquivo define a função *fobjTE* que recebe como entrada a solução que se deseja avaliar e uma matriz com o tempo que cada máquina demora para executar cada tarefa (estes tempos são carregados do arquivo `i5x25.mat` disponibilizado pelo professor). Pela multiplicação vetorial de cada linha da matriz dos tempos com cada coluna da matriz x (solução), tem-se o tempo de operação de cada máquina. A avaliação da solução na função objetivo é, como já visto, o maior dentre os tempos de operação das máquinas.

Antes de implementar o SA propriamente dito, foi necessário também criar funções que geram novas soluções em dada vizinhança. Para este trabalho, duas funções de vizinhança foram criadas. A primeira, para uma dada solução x, gera uma nova solução y trocando aleatoriamente as máquinas que executam n tarefas (n também é um parâmetro da função), e está no arquivo `neighbor1TE.m`. Repare que aqui não ocorre necessariamente uma troca entre máquinas. A outra função de vizinhança recebe uma solução x e gera uma nova solução y escolhendo duas linhas aleatoriamente (duas tarefas), e trocando-as, de forma que duas máquinas trocam as tarefas entre si. Está no arquivo `neighbor2TE.m`.

II. CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENT

The authors would like to thank...

REFERÊNCIAS

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.