

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 .*

II. DESENVOLVIMENTO

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:

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

O objetivo então se torna:

$$\min C_{max}$$

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

sujeito a:

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

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

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

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:

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

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:

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

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

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

```

1 Defina um contador  $k = 0$ ;
2 Defina uma temperatura inicial  $t_k \geq 0$ ;
3 Defina  $T_k$  (função que controla a variação da temperatura);
4 Defina  $M_k$  (no. de iterações executadas na temperatura  $t_k$ );
5 Selecione uma solução inicial  $\mathbf{x} \in \Omega$ ;
6 while critério de parada não alcançado do
7   Defina o contador  $m = 0$ ;
8   while  $m \leq M_k$  do
9     Gere uma solução  $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ ;
10    Calcule  $\Delta E = f(\mathbf{x}') - f(\mathbf{x})$ ;
11    if  $\Delta E \leq 0$  then
12       $\mathbf{x} \leftarrow \mathbf{x}'$ ;
13    else
14       $\mathbf{x} \leftarrow \mathbf{x}'$  com probabilidade  $\exp(-\Delta E/t_k)$ ;
15     $m \leftarrow m + 1$ ;
16   $t_{k+1} \leftarrow T_k(t_k)$ ;
17   $k \leftarrow k + 1$ ;

```

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`. Com estas funções de vizinhança, a restrição de que cada linha pode ter apenas um número 1 (cada tarefa só pode ser executada por uma máquina) ainda é atendida.

O algoritmo de otimização foi implementado no arquivo `minTempoEntrega.m`, que tem a função de mesmo nome. Utiliza a estratégia do *Simulated Annealing* e também, como auxílio, todos os outros algoritmos apresentados até aqui para o problema em questão. A função `minTempoEntrega` implementada no arquivo possui dois argumentos, o que vai facilitar, posteriormente, no ajuste de parâmetros do método implementado. Este ajuste será apresentado na seção de resultados.

2) *Minimização da Soma Ponderada dos Atrasos e Adiantamentos*: TODO

C. Resultados:

Nesta sessão serão apresentados os resultados dos algoritmos.

1) *Minimização do Tempo Total de Entrega*: Como já mencionado, o algoritmo de otimização foi baseado no *Simulated Annealing*. Esta abordagem exige o ajuste de alguns parâmetros, dentre eles o multiplicador α de temperatura, que é um valor entre 0 e 1, que vai diminuir gradualmente a temperatura do algoritmo, e consequentemente diminuir a probabilidade de se aceitar um movimento de piora na busca pela melhor solução. Outro parâmetro que deve ser ajustado é a quantidade n

de tarefas que terão a máquina aleatoriamente trocada, na primeira função de vizinhança. Os dados para execução do problema foram disponibilizados pelo professor, são carregados pelo arquivo `i5x25.mat` e podem ser vistos na tabela da Figura 2.

Tarefa	Máquina					Peso
	1	2	3	4	5	
1	2	1	4	7	8	8
2	8	3	2	1	5	5
3	8	8	8	4	1	7
4	4	9	10	4	5	10
5	9	10	7	5	3	2
6	3	3	4	3	8	5
7	9	1	1	8	3	2
8	10	6	4	9	6	8
9	9	8	1	1	9	10
10	6	1	4	10	6	6
11	10	10	6	5	9	3
12	4	7	6	2	6	7
13	9	5	3	6	2	2
14	4	7	3	8	1	7
15	2	9	10	8	6	2
16	5	8	2	6	9	10
17	7	8	7	1	8	1
18	6	9	1	8	9	1
19	1	5	8	8	10	6
20	3	2	7	9	4	1
21	1	6	7	9	10	1
22	10	8	4	4	9	9
23	6	2	9	3	8	5
24	2	1	1	6	5	10
25	1	9	3	10	8	3
DueDate	6					

Figura 2. Tabela de dados para execução dos algoritmos.

Para demonstrar os efeitos da temperatura (parâmetro α), uma primeira instancia foi executada, utilizando como parâmetros $\alpha = 0.5$ e $n = 3$. A Figura 3 mostra um primeiro resultado desta execução, plotando a avaliação da solução aceita por cada iteração. Como se pode notar, no início do algoritmo, muitas soluções de piora são aceitas, e aos poucos são aceitos cada vez mais apenas movimentos de melhora. Esta execução passou por 2141 iterações e a melhor solução encontrada possui avaliação na função objetivo igual a 20.

Executando mais uma vez, mas agora para $\alpha = 0.1$ e $n = 3$, obtemos o resultado mostrado na Figura 4. Como se pode notar, agora menos movimentos de piora são aceitos nas iterações iniciais. Neste caso foram executadas 2488 iterações, com um valor ótimo igual a 14. Ao custo de mais iterações, obteve-se um ponto de ótimo local melhor.

É necessário então escolher quais serão os parâmetros utilizados, e ainda qual será a função de vizinhança usada. Para tanto, um script de execução foi criado e está no arquivo `multirunTE.m`. Neste, o código é executado uma quantidade de vezes desejada, e os valores de ótimo e quantidade de iterações para cada execução são plotados. Ajustando o algoritmo para $\alpha = 0.5$ e $n = 3$,

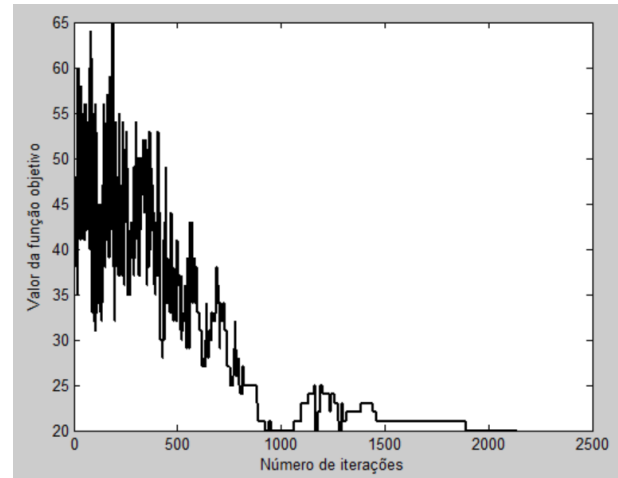


Figura 3. Primeiro resultado para execução com $\alpha = 0.5$ e $n = 3$.

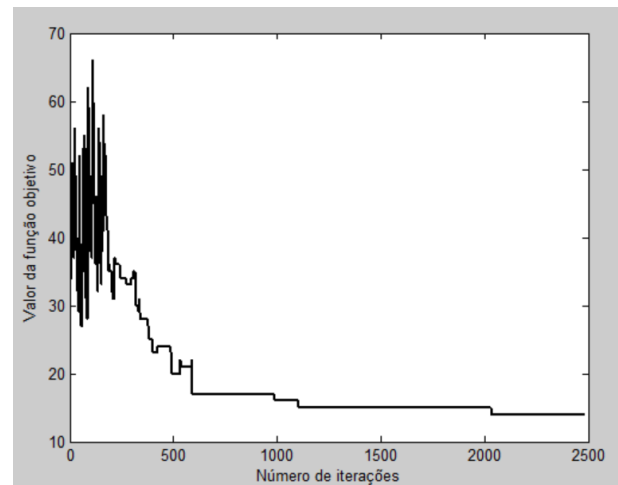


Figura 4. Primeiro resultado para execução com $\alpha = 0.1$ e $n = 3$.

após 100 execuções obtemos o resultado mostrado na Figura 5. A linha preta no meio dos gráficos representa a média dos valores. Encontrou-se valor ótimo médio igual a 17,83 e valor médio da quantidade de iterações igual a 2209,1.

Foi experimentado também a execução do algoritmo com $\alpha = 0.1$ e $\alpha = 0.01$, ambos mantendo $n = 3$. Os resultados podem ser vistos nas Figuras 6 e 7, respectivamente. Para o primeiro caso, a média do valor ótimo foi 14,87 (com mínimo em 12 e máximo em 21) e a média da quantidade de iterações foi 2253,6 (com mínimo em 2001 e máximo em 2623). Para o segundo caso, a média do valor ótimo foi 15,95 (com mínimo em 13 e máximo em 19) e a média da quantidade de iterações foi 2509,4 (com mínimo em 2299 e máximo em 2581). Optou-se então por manter o algoritmo ajustado a $\alpha = 0.1$, por ter uma média de valor ótimo alcançado melhor.

Como todos estes resultados foram obtidos executando

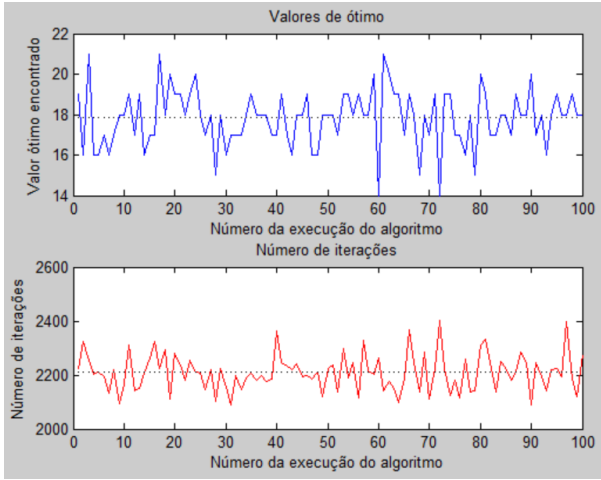


Figura 5. 100 execuções com $\alpha = 0.5$ e $n = 3$.

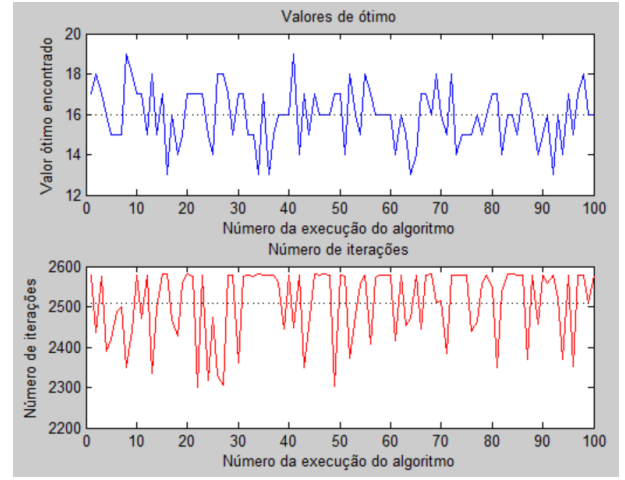


Figura 7. 100 execuções com $\alpha = 0.01$ e $n = 3$.

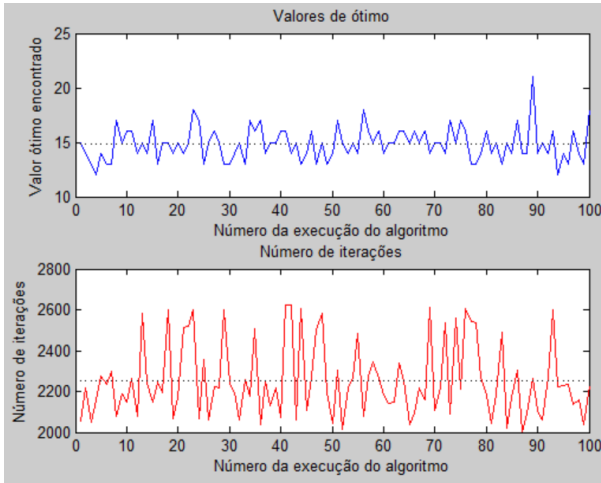


Figura 6. 100 execuções com $\alpha = 0.1$ e $n = 3$.

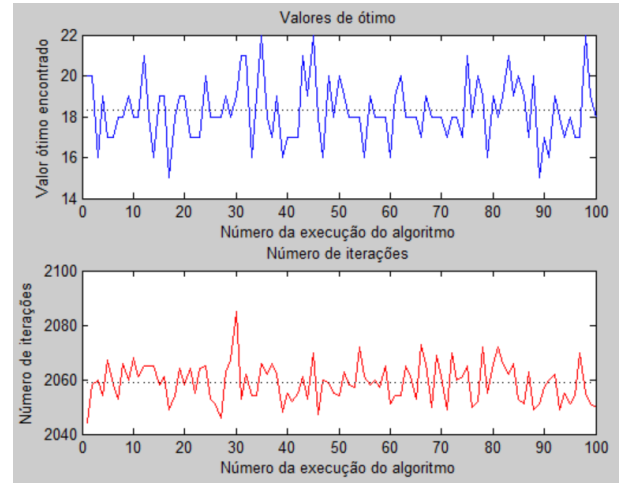


Figura 8. 100 execuções com $\alpha = 0.1$ e $n = 5$.

o algoritmo com a primeira forma de vizinhança (*neighborITE*), precisamos também decidir um valor para n . Mantendo $\alpha = 0.1$, *multirunTE.m* foi executado para $n = 5$ e $n = 1$. Os resultados podem ser vistos nas Figuras 8 e 9, respectivamente. Para o primeiro caso, a média do valor ótimo foi 18,29 (com mínimo em 15 e máximo em 22) e a média da quantidade de iterações foi 2059 (com mínimo em 2044 e máximo em 2085). Para o segundo caso, a média do valor ótimo foi 14,77 (com mínimo em 12 e máximo em 20) e a média da quantidade de iterações foi 2260,8 (com mínimo em 2003 e máximo em 2622). Escolhemos então manter $n = 1$.

Agora é preciso avaliar a segunda função de vizinhança, *neighbor2TE*. Para este caso, o parâmetro n não tem interferência, e α será mantido em 0.1. Mais uma vez com a ajuda do script *multirunTE.m*, o algoritmo foi executado mais 100 vezes e o resultado pode ser visto na Figura 10. A média do valor ótimo foi

18,86 (com mínimo em 12 e máximo em 41) e a média da quantidade de iterações foi 2131,1 (com mínimo em 2003 e máximo em 2308). Como para este caso o valor de ótimo encontrado varia muito e chega a altos valores, escolheu-se manter a função de vizinhança *neighborITE*.

Com todos os ajustes feitos, o algoritmo foi executado mais 5 vezes, e o resultado sumarizado pode ser visto na Figura 11. Os resultados numéricos são apresentados na Tabela 1.

III. CONCLUSÃO

LEMBRAR DE FALAR QUE O SA É BOM PARA OBTEN BONS ÓTIMOS LOCAIS, NÃO NECESSARIAMENTE ENCONTRANDO O ÓTIMO GLOBAL.

ACKNOWLEDGMENT

The authors would like to thank...

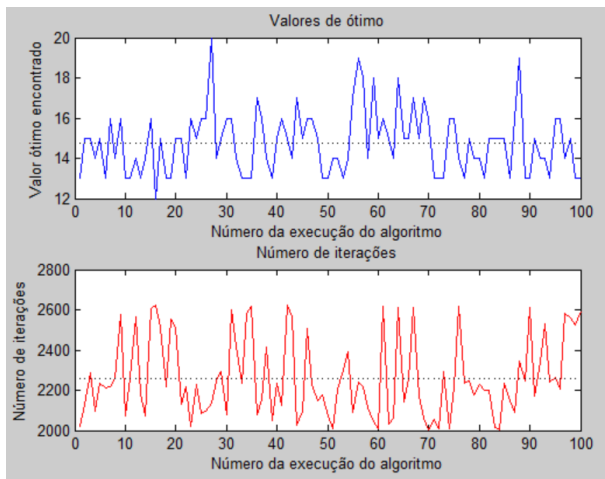


Figura 9. 100 execuções com $\alpha = 0.1$ e $n = 1$.

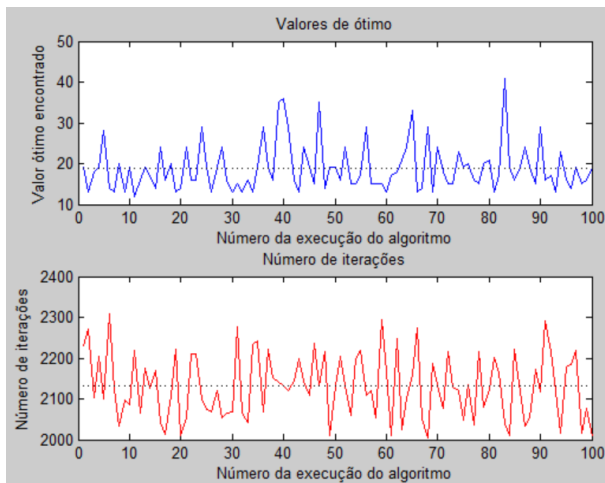


Figura 10. 100 execuções com $\alpha = 0.1$ e vizinhança *neighbor2TE*.

REFERÊNCIAS

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

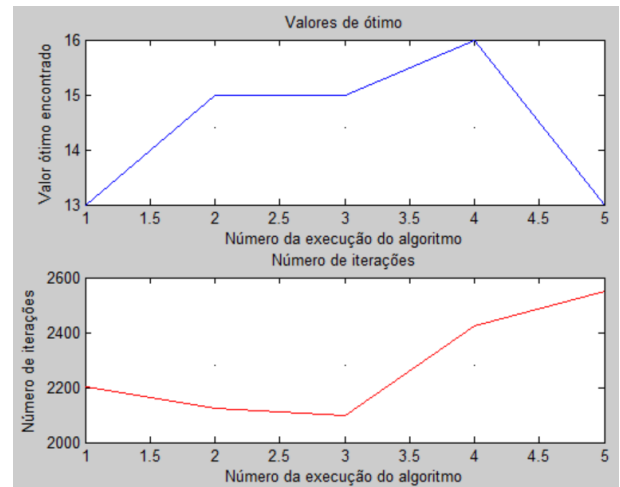


Figura 11. Resultado final para 5 iterações - Minimização do Tempo de Entrega.

Ótimo Encontrado	Iterações
13	2204
15	2124
15	2097
16	2424
13	2549

Tabela I

RESULTADOS NUMÉRICOS - MINIMIZAÇÃO DO TEMPO DE ENTREGA.