Teoria da Decisão

Introdução às Metaheurísticas

Prof. Lucas S. Batista

lusoba@ufmg.br

www.ppgee.ufmg.br/∼lusoba

Universidade Federal de Minas Gerais
Escola de Engenharia
Graduação em Engenharia de Sistemas

Sumário

Processo de Otimização

- Frequentemente, o benefício ou o custo (i.e., o efeito) da implantação de uma solução $\boldsymbol{x}$ pode ser expresso por meio de uma função $f(\cdot)$ de variáveis de decisão, onde $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$.

- Então, determinar o melhor arranjo dessas variáveis que minimiza ou maximiza essa função mérito consiste em um *processo de otimização*.

Processo de Otimização

Otimização

Processo que utiliza métodos computacionais para encontrar a "melhor" forma de projetar e/ou operar um dado sistema, representada pela melhor combinação de valores para as variáveis do problema, considerando seus objetivos e suas restrições de projeto e de operação.

O processo de otimização sempre resulta, ou busca justificação, em um impacto econômico, representado por:

- qualidade do produto, custo da produção, competitividade.

Processo de Otimização

Otimizar

Significa minimizar (ou maximizar) uma dada função:

$$\text{Encontrar } \boldsymbol{x} \in \mathcal{F} : f(\boldsymbol{x}) \leq f(\boldsymbol{y}),\ \forall\, \boldsymbol{y} \in \mathcal{F}$$

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}),\ \boldsymbol{x} \in \mathcal{F}$$

Problema de Otimização Mono-objetivo

- Formulação geral:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \in \mathbb{R},\ \boldsymbol{x} \in \mathcal{F}$$

$$\mathcal{X} = \{\boldsymbol{x} = (x_1, x_2, \ldots, x_n),\ x_i \in \mathcal{D}_i\}$$

$$\mathcal{F} = \begin{cases} g_i(\boldsymbol{x}) \leq 0; & i = 1, \ldots, p \\ h_j(\boldsymbol{x}) = 0; & j = 1, \ldots, q \\ \boldsymbol{x} \in \mathcal{X} \end{cases}$$

Problema de Otimização Mono-objetivo

- Caso particular (programação linear):

$$\min_{\boldsymbol{x}} \boldsymbol{cx} \in \mathbb{R}, \ \boldsymbol{x} \in \mathcal{F}$$

$$\mathcal{F} = \begin{cases} \sum_k a_{ik} x_k \leq 0; & i = 1, \ldots, p \\ \sum_k b_{jk} x_k = 0; & j = 1, \ldots, q \\ x_k \geq 0 \end{cases}$$

Sumário

Motivação

- Todos os dias, engenheiros e tomadores de decisão são confrontados com problemas de crescente complexidade.
- Esses problemas emergem de diversos setores técnicos:
  - projeto de circuitos elétricos/eletrônicos;
  - projeto de sistemas de controle;
  - projeto de sistemas mecânicos;
  - processamento de imagens;
  - pesquisa operacional.
- Esses desafios podem, frequentemente, ser modelados como *problemas de otimização*.

Otimização “Difícil”

- Dois tipos de problemas de otimização são claramente postos: problemas “discretos” e problemas com variáveis contínuas.
- Problemas discretos: caixeiro viajante, roteamento de veículos.
- Problemas contínuos: máquinas elétricas, controladores PI.
- Problemas mistos: envolvem simultaneamente variáveis discretas e contínuas.
- Essa diferenciação é útil para definir o domínio de “dificuldade” do problema de otimização.

Otimização “Difícil”

Problemas discretos “difíceis”

Não se conhece um algoritmo *polinomial* exato. Este é o caso, particularmente, dos problemas “NP-difíceis”.

Problemas contínuos “difíceis”

Não se conhece um algoritmo exato capaz de localizar o ótimo global em um número finito de iterações.

Otimização "Difícil"

Muito esforço foi dedicado, separadamente, à solução desses problemas:

No campo dos problemas contínuos "difíceis"...

Existe um arcabouço significativo de métodos tradicionais para *otimização global*. Entretanto, sua efetividade depende de propriedades específicas do problema, e.g., diferenciabilidade, convexidade, modalidade.

No campo dos problemas discretos "difíceis"...

Existe um arsenal de *heurísticas*, as quais encontram soluções próximas do ótimo. Entretanto, a maioria delas é concebida para um problema específico.

Otimização “Difícil”

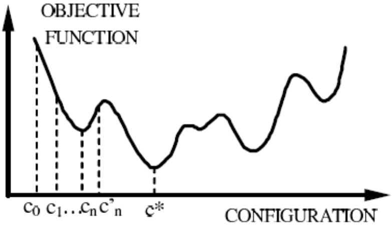
- A chegada das *Metaheurísticas* (MH) marca uma reconciliação de ambos os domínios.
- De fato, elas são aplicadas a todos os tipos de problemas discretos e podem ser adaptadas a problemas contínuos.
- Possuem em comum as seguintes características:
  - são estocásticas;
  - possuem origem discreta e mesmo em problemas contínuos não exigem diferenciabilidade, convexidade, modalidade;
  - são inspiradas em analogias físicas (SA), biológicas (TS, EAs) ou etológicas (ACO, PSO);
  - compartilham as mesmas desvantagens, i.e., ajuste de parâmetros e alto custo computacional.

Otimização “Difícil”

- Em geral, é impossível assegurar com certeza a efetividade de uma dada MH quando aplicada a um dado problema.
- As MHs podem ser facilmente estendidas para:
  - otimização multiobjetivo;
  - otimização multimodal;
  - otimização dinâmica;
  - implementações paralelas.

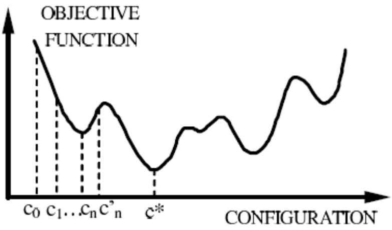
Limitação Geral de Métodos Clássicos

- Métodos clássicos, ou métodos de decida, aceitam somente "movimentos" de melhora (possuem convergência monotônica).
- Esses algoritmos de aperfeiçoamento iterativo não conduzem, em geral, ao ótimo global, mas a um mínimo local específico (c_n).



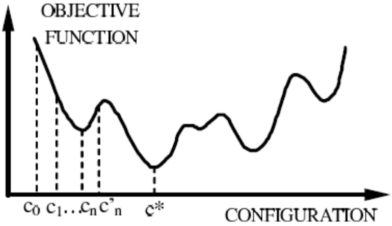
Limitação Geral de Métodos Clássicos

- Para melhorar a efetividade dos métodos clássicos, eles podem ser aplicados repetidas vezes, partindo de configurações iniciais distintas.
- Esse processo, entretanto, aumenta o custo computacional do algoritmo, não garante a determinação do ótimo c^* e torna-se inefetivo com o aumento do número de mínimos locais.

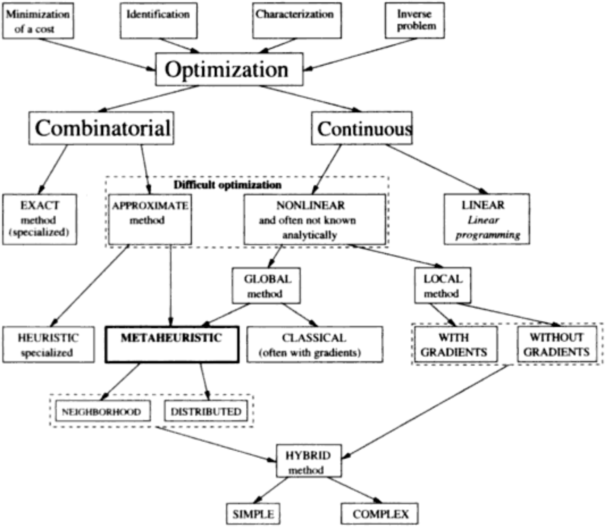


Fonte de Efetividade das MHs

- MHs são capazes de escapar de ótimos locais!!!
- MHs baseadas em "vizinhança" (SA, VNS, ILS) aceitam a *degradação* temporária da solução, permitindo encontrar c_n, c_n' e c^*.
- MHs "distribuídas" (EAs) *evoluem* paralelamente uma população de soluções candidatas e empregam estratégias específicas para exploração do espaço de busca.



Classificação Geral dos Métodos de Otimização Mono-objetivo



Sumário

Introdução

- *Variable Neighborhood Search* (VNS) é uma metaheurística proposta por *N. Mladenovic, P. Hansen (1997)*.
- Baseia-se na ideia de uma mudança sistemática de vizinhança:
  - na *fase de refinamento*, para encontrar um ótimo local; e
  - na *fase de perturbação*, para escapar de bacias de atração.

Variable Neighborhood Descent

Algoritmo 1: Variable Neighborhood Descent

```
// Função VND(x, k_max)
k ← 1;
repeat
    x' ← arg min_{y∈N_k(x)} f(y) ;   /* encontre o "melhor" vizinho em N_k(x) */
    x, k ← NeighborhoodChange(x, x', k) ;   /* altere a vizinhança */
until k > k_max;
Retorne x;
```

- No VND, a mudança de vizinhança é realizada de maneira determinística.
- Frequentemente, $k_{max} \leq 2$ em heurísticas de busca local.

Neighborhood Change

Algoritmo 2: Neighborhood Change

```
// Função NeighborhoodChange(x, x', k)
if f(x') < f(x) then
    x ← x' ;                          /* atualize solução */
    k ← 1 ;                           /* vizinhança inicial */
else
    k ← k + 1 ;                       /* próxima vizinhança */
Retorne x, k;
```

Reduced VNS (RVNS)

Algoritmo 3: Reduced VNS

```
// Função RVNS (x, k_max, t_max)
repeat
    k ← 1;
    repeat
        x' ← Shake(x, k);
        x, k ← NeighborhoodChange(x, x', k);
    until k > k_max;
    t ← CpuTime( );
until t > t_max;
Retorne x;
```

- No RVNS, soluções aleatórias são selecionadas a partir de $\mathcal{N}_k(\boldsymbol{x})$.
- Uma técnica de refinamento não é aplicada.

Basic VNS (BVNS)

- O método BVNS combina mudanças de vizinhança determinística e estocástica.
- A determinística é representada por uma heurística de busca local.
- A estocástica é representada por uma seleção aleatória de uma solução da k-ésima estrutura de vizinhança.

Heurística de Busca Local - *Best Improvement*

Algoritmo 4: Best Improvement heuristic

// **Função** BestImprovement($\boldsymbol{x}$)

repeat

  $\boldsymbol{x}' \leftarrow \boldsymbol{x}$;

  $\boldsymbol{x} \leftarrow \arg \min_{\boldsymbol{y} \in \mathcal{N}(\boldsymbol{x})} f(\boldsymbol{y})$;

until $f(\boldsymbol{x}) \geq f(\boldsymbol{x}')$;

Retorne $\boldsymbol{x}$;

Heurística de Busca Local - *First Improvement*

Algoritmo 5: First Improvement heuristic

// **Função** `FirstImprovement(`$\boldsymbol{x}$`)`

repeat
    $\boldsymbol{x}' \leftarrow \boldsymbol{x}$;
    $i \leftarrow 0$;
    repeat
        $i \leftarrow i + 1$;
        $\boldsymbol{x} \leftarrow \arg\min\{f(\boldsymbol{x}), f(\boldsymbol{x}^i)\},\ \boldsymbol{x}^i \in \mathcal{N}(\boldsymbol{x})$;
    until $f(\boldsymbol{x}) < f(\boldsymbol{x}')$ ***or*** $i = |\mathcal{N}(\boldsymbol{x})|$;
until $f(\boldsymbol{x}) \geq f(\boldsymbol{x}')$;
Retorne $\boldsymbol{x}$;

Basic VNS (BVNS)

Algoritmo 6: Basic VNS

```
// Função BVNS (x, k_max, t_max)
repeat
    k ← 1;
    repeat
        x' ← Shake(x, k);
        x'' ← BestImprovement(x');
        x, k ← NeighborhoodChange(x, x'', k);
    until k > k_max;
    t ← CpuTime( );
until t > t_max;
Retorne x;
```

General VNS (GVNS)

- O método GVNS combina as técnicas VNS e VND.
- Esta estratégia está relacionada às aplicações de maior sucesso citadas na literatura sobre VNS.

Algoritmo 7: General VNS

// **Função** GVNS ($\boldsymbol{x}, \ell_{max}, k_{max}, t_{max}$)

repeat
    $k \leftarrow 1$;
    repeat
        $\boldsymbol{x}' \leftarrow Shake(\boldsymbol{x}, k)$;
        $\boldsymbol{x}'' \leftarrow VND(\boldsymbol{x}', \ell_{max})$;
        $\boldsymbol{x},\ k \leftarrow NeighborhoodChange(\boldsymbol{x}, \boldsymbol{x}'', k)$;
    until $k > k_{max}$;
    $t \leftarrow CpuTime(\ )$;
until $t > t_{max}$;
Retorne $\boldsymbol{x}$;

Sumário

Exemplos de Estruturas de Vizinhança

- Suponha um problema de sequenciamento de máquinas paralelas não relacionadas com tempos de preparação.

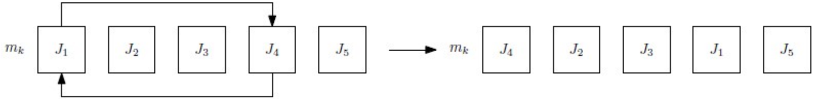
- Quais estruturas de vizinhança podem ser empregadas?

Exemplos de Estruturas de Vizinhança

- Shift
- Realocação de uma tarefa para outra posição da mesma máquina.

Exemplos de Estruturas de Vizinhança

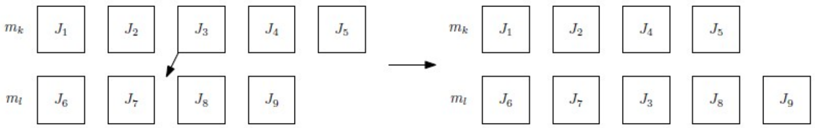
- Switch
- Troca a posição de duas tarefas processadas na mesma máquina.



Exemplos de Estruturas de Vizinhança

- Task Move

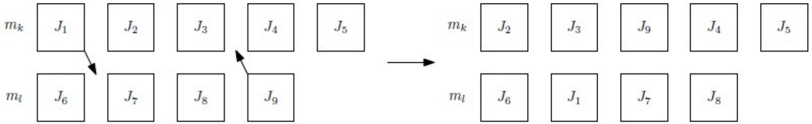
- Movimentação de uma tarefa de uma máquina de origem para uma outra máquina.



Exemplos de Estruturas de Vizinhança

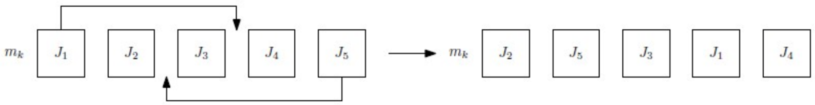
- Swap

- Realocação de duas tarefas aleatórias entre duas máquinas.



Exemplos de Estruturas de Vizinhança

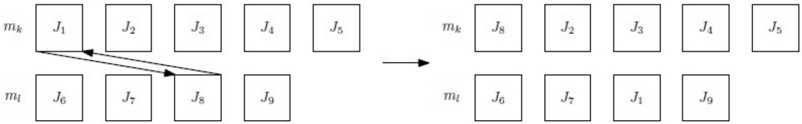
- Two-Shift
- Desloca a posição de duas tarefas processadas em uma mesma máquina.



Exemplos de Estruturas de Vizinhança

- Direct Swap

- Troca de duas tarefas entre duas máquinas, mantendo as posições originais nessas máquinas.



Exemplos de Estruturas de Vizinhança

Vizinhança	Máquinas	Cardinalidade*	Complexidade*	Altera *Makespan*
Shift	1	$\frac{n^2 - nm}{m}$	$O(n^2)$	$\frac{1}{m}$
Switch	1	$\frac{-n^2}{2m} - \frac{n}{2}$	$O(n^2)$	$\frac{1}{m}$
Task Move	2	$\frac{-n^2}{m} + n^2 + nm - n$	$O(n^2)$	$\frac{2}{m}$
Swap	2	$\frac{n^4(m-1)}{2m^3}$	$O(n^4)$	$\frac{2m-2}{m^2-m}$
Direct Swap	2	$\frac{n^2(m-1)}{2m}$	$O(n^2)$	$\frac{2m-2}{m^2-m}$
Two-Shift	1	$\frac{n^4 - 2n^3m + n^2m^2}{2m^3}$	$O(n^4)$	$\frac{1}{m}$

* n é o número de tarefas e m é o número de máquinas acessíveis.

Literatura Especializada

M. Gendreau, J.-Y. Potvin (eds.), Handbook of Metaheuristics, Springer, 2nd ed., 2010.

J. Dréo, P. Siarry, A. Pétrowski, E. Taillard, Metaheuristics for Hard Optimization: Methods and Case Studies, Springer, 2006.

L.M. Pereira, Análise de estruturas de vizinhança para o problema de sequenciamento de máquinas paralelas não relacionadas com tempos de preparação, Dissertação de Mestrado, PPGEE/UFMG, 2019.

Início