Prof. Lucas de Souza Batista - DEE/EE/UFMG

Otimização de Redes

Heurísticas Matemáticas:
Uma Introdução ao Tema

Heurísticas Matemáticas

- (Meta)heurísticas hibridizadas com métodos exatos;
- Algoritmos híbridos normalmente se baseiam em uma estrutura “master-slave”:
  - (i) Método exato controla o uso da MH; ou
  - (ii) MH controla as chamadas ao método exato;

Heurísticas Matemáticas

- Algoritmos híbridos do tipo (i):
  - MH é embutida em um *solver;*
  - *MIP solvers* podem consumir muito tempo até encontrar a primeira solução viável;
  - *Solvers B&C modernos* já exploram o potencial de MHs para determinar rapidamente boas soluções iniciais (estágios iniciais de exploração da árvore);
  - *Limitantes* relacionados são usados para podar ramos da árvore (acelera o processo de busca e reduz custo).

Heurísticas Matemáticas

- Algoritmos híbridos do tipo (ii):
  - a *definição da vizinhança* segue a lógica de uma MH
  - a *exploração da vizinhança* é realizada por um método exato

Introdução

- Algoritmos híbridos podem ser de natureza *exata* ou *heurística*;
  - *Métodos exatos* garantem soluções ótimas quando tempo computacional suficiente é fornecido;
- *Heurísticas* objetivam somente encontrar boas soluções aproximadas em um tempo mais restritivo (convergência não é garantida);
- A maioria das técnicas híbridas são de natureza heurística, e *métodos de programação matemática são usados para impulsionar o desempenho de uma MH*.

Introdução

- A maioria das técnicas exatas para solução de *combinatorial optimization problems* (COPs) se baseia na *busca em árvore*:
  - o espaço de busca é particionado recursivamente fixando algumas variáveis ou impondo restrições adicionais.

Introdução

- No *B&B*, limitantes superiores e inferiores são determinados para os valores objetivos das soluções:
  - toda solução viável provê um limitante superior;
  - subespaços cujos limitantes inferiores excedem os limitantes superiores são descartados;
- MHs são viáveis para encontrar rapidamente soluções aproximadas, úteis no processo de poda do *B&B*.

Introdução

- Relaxações:
  - várias ou todas as restrições de um problema são relaxadas ou omitidas;
  - muito usadas para obter problemas relacionados, simples, que podem ser resolvidos *eficientemente* fornecendo limitantes e soluções aproximadas (não necessariamente factíveis) para o problema original;

Introdução

- Relaxação LP de um IP:
  - $z_{IP} = min\{cx \mid Ax \geq b, x \geq 0, x \in Z^n\}$
  - $z_{LP} = min\{cx \mid Ax \geq b, x \geq 0, x \in R^n\}$
  - grandes instâncias de LPs podem ser resolvidas eficientemente via Simplex ou Pontos Interiores;
  - a solução do LP provê um limitante inferior para o IP, i.e., $z_{LP} \leq z_{IP}$.

Alg. Híbridos Tipo (i)

Definição de Limitantes com MHs

- MHs podem ser aplicadas ao problema original antes do início do *B&B*:
  - provê soluções factíveis iniciais (CPLEX, GUROBI);
  - reduz espaço de busca do método exato;
  - acelera o processo de otimização;
  - podem ser aplicadas repetidas vezes ao longo da busca na árvore;
  - muitas aplicações podem consumir muito tempo!

Solution Merging

- Novas soluções, possivelmente melhores, são criadas a partir dos atributos de outras soluções promissoras (integração entre *B&B* e EA):
  - *Mutation & Recombination* (fixa variáveis comuns e otimiza as demais com um MIP *solver*);
  - *Path relinking;*
  - *into CPLEX since version 10!*

Alg. Híbridos Tipo (ii)

Heurísticas Matemáticas

- Tema Abordado:
  - Uso de ***algoritmos exatos*** para melhorar o desempenho de ***métodos de busca local estocásticos***.
  - Foca em algoritmos cujo ***framework principal é baseado em busca local,*** mas usa métodos exatos para resolver subproblemas.

Introdução

- *Combinatorial optimization problems* (COPs) são intrigantes pois apesar da **facilidade de definição**, são muito difíceis de resolver (***NP*-difícil**).
- Esta dificuldade, aliada a sua **enorme importância prática**, tem motivado o desenvolvimento de inúmeras técnicas de solução para os mesmos.
- Estas técnicas de solução podem ser classificadas como:
  - algoritmos exatos;
  - algoritmos aproximativos.

Introdução

- Algoritmos Exatos:
  - Garantem a determinação de uma solução ótima;
  - Prova sua otimalidade para toda instância de tamanho finito (tempo de execução finito);
  - Caso contrário, prova que não existe solução viável;
  - ***Integer Programming*** **(IP):** *B&B, B&C, B&P, DP*

Introdução

- Vantagens de Algoritmos Exatos:
  - Provam otimalidade (se o método converge);
  - Fornecem informações valiosas sobre limitantes superior/inferior da solução ótima;
  - Permitem a poda de partes do espaço de busca onde não existem soluções ótimas;
  - Softwares comerciais: GLPK, CPLEX, GUROBI

Introdução

- Desvantagens de Algoritmos Exatos:
  - Para muitos problemas, o **tamanho** de instâncias resolvidas na prática é muito limitado;
  - O **consumo de memória** por algoritmos exatos pode ser muito alto, podendo causar a parada prematura do programa;
  - Algoritmos de alto desempenho são **específicos** para cada problema (IP requer muito tempo de desenvolvimento);
  - A **extensão** é frequentemente difícil mesmo para variantes de um mesmo problemas.

Introdução

- Algoritmos Aproximativos:
  - Se soluções ótimas não podem ser obtidas de forma eficiente, abre-se mão da ***otimalidade*** em troca de ***eficiência computacional***.
- Soluções ***aproximadas*** podem ser encontradas em *tempo razoavelmente curto*.
- ***Stochastic Local Search*** **(SLS):** VNS, GRASP, ILS

Introdução

- Vantagens de Algoritmos Aproximativos:
  - Representam os métodos de melhor desempenho para uma grande variedade de COPs;
  - Podem examinar um grande número de soluções candidatas em um curto tempo computacional;
  - São frequentemente mais fáceis de adaptar a variantes de um problema, i.e., mais flexíveis;
  - São mais fáceis de entender e implementar do que métodos exatos.

Introdução

- Desvantagens de Algoritmos Aproximativos:
  - Não provam otimalidade, nem fornecem limitantes para a qualidade das soluções;
  - Usualmente, não podem reduzir o espaço de busca;
  - Não possuem critérios de parada bem definidos;
  - Frequentemente, enfrentam dificuldades com problemas fortemente restritos (regiões factíveis desconexas);
  - SLS Solvers disponíveis não são, em geral, eficientes.

Introdução

- Métodos IP e SLS possuem vantagens e limitações;
- Podem ser vistos como complementares;
- Podem ser combinados em poderosos algoritmos:
  - heurísticas matemáticas!!!
  - Exemplo mais usual:

    SLS —> determinar bons limitantes superiores iniciais

    IP —> podar soluções inferiores

Introdução

- Escopo*:
  - Algoritmo principal (*the master*) —> SLS
  - Método IP —> otimização de subproblemas

**A filosofia inversa é discutida no livro texto.*

Introdução

- Escopo (a partir de várias estratégias híbridas da literatura):
  - Uso de IP para explorar grandes vizinhanças em SLS;
  - Formas de aperfeiçoar SLS resolvendo subproblemas de maneira exata;
  - Uso de técnicas de B&B para aperfeiçoar SLS Construtivas;
  - Refinamento da estrutura de boas soluções encontradas por SLS;
  - Refinamento de informações de relaxações em SLS.

Explorando Grandes Vizinhanças

- ***Em uma busca local,*** melhores soluções têm mais chances de serem obtidas em ***grande vizinhanças*** do que em ***vizinhanças pequenas/simples;***
- Entretanto, uma busca sobre grandes vizinhanças pode exigir um ***tempo computacional considerável;***
- Além disso, muitas vizinhanças grandes possuem ***crescimento exponencial*** com o tamanho da instância.

Explorando Grandes Vizinhanças

- Métodos SLS para grandes vizinhanças podem ser divididos em duas classes:
  - métodos de busca heurísticos
    - *variable-depth search algorithms*
    - *ejection chains*
  - métodos de busca exatos
    - o problema de busca é definido como um *problema de otimização,* que é resolvido de forma exata.

Explorando Grandes Vizinhanças

- Definição do problema de busca em vizinhança #**1**:
  - a exploração da vizinhança *completa* é modelada como um problema de otimização;
  - a tarefa do *algoritmo exato* é realizar a otimização do *neighborhood search problem* (NSP).

Explorando Grandes Vizinhanças

Algorithm 4.2.1 *Neighborhood search*

Step 1: *Initialization*

Let s be a feasible initial solution.

Step 2: *Local search*

while `stopping_criterion` is not met **do**

Define a search problem $\mathcal{P}(s)$ that depends on s.

Find $opt(\mathcal{P}(s))$, an optimal solution of $\mathcal{P}(s)$.

Let s' be the solution induced by $opt(\mathcal{P}(s))$.

if s' is better than s w.r.t. the objective function **then** $s = s'$.

enddo

Step 3: Return s.

Explorando Grandes Vizinhanças

- Definição do problema de busca em vizinhança #**2**:
  - em cada passo da busca local, uma parte da solução corrente *s* é mantida fixa (*solução parcial*);
  - os valores das demais variáveis de decisão continuam "livres", os quais são arranjados de forma ótima;
  - a tarefa do *algoritmo exato* é realizar a otimização do *partial neighborhood search problem* (PNSP).

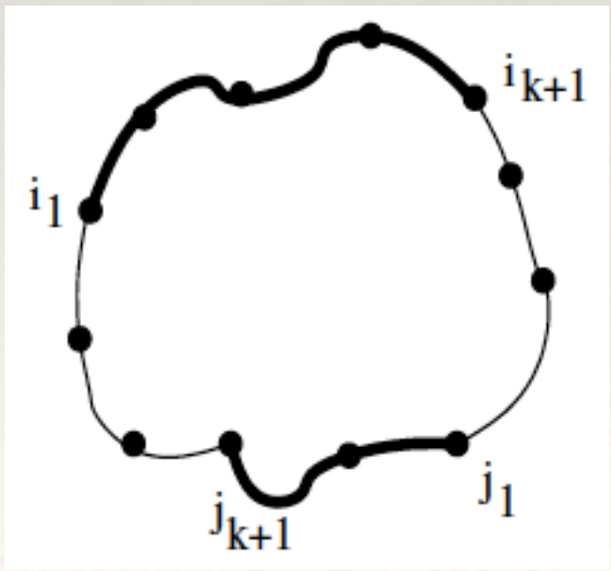
Explorando Grandes Vizinhanças

Algorithm 4.2.2 *Partial neighborhood search*

Step 1: *Initialization*
Let s be an initial solution.
Step 2: *Neighborhood search*
while `improvement found` **do**
Step 3: *Neighborhood scan*
while not full neighborhood examined **do**
Delete solution components from solution s resulting in a partial solution: $s_p = s \setminus r$.
Define a search problem $\mathcal{P}(r)$.
Find $opt(\mathcal{P}(r))$, the optimal solution of $\mathcal{P}(r)$.
Perform the move induced by $opt(\mathcal{P}(r))$, resulting in solution $\bar{s}'$; $s' = s_p \otimes \bar{s}'$.
if s' is better than s w.r.t. the objective function **then** $s = s'$.
enddo
enddo
Step 4: Return s.

Explorando Grandes Vizinhanças

- **Exemplo PNSP: *Hyperopt Neighborhoods* (TSP)**
  - *hyperopt neighborhood* é baseada na noção de hiper-arestas;
  - uma hiper-aresta é definida como um subcaminho de uma rota;



Explorando Grandes Vizinhanças

- **Exemplo PNSP: *Hyperopt Neighborhoods* (TSP)**
  - uma hiper-aresta entre os nós *i* e *j* é dada por *H(i,j)*;
  - o tamanho de uma hiper-aresta é dado pelo número de arestas existentes no subcaminho;
  - a vizinhança *k-hyperopt* consiste de todos os movimentos *k-hyperopt;*

Explorando Grandes Vizinhanças

- **Exemplo PNSP: *Hyperopt Neighborhoods* (TSP)**
  - cada subproblema (definido pelo conjunto de hiper-arestas) é resolvido de forma ótima usando movimentos *k-hyperopt*;
  - estratégia eficiente somente quando $k < 3$; caso contrário, sugere-se programação dinâmica;
  - as soluções parciais compõem uma nova solução final, a qual é comparada à solução incumbente.

Explorando Grandes Vizinhanças

- **Outras Estratégias:**
- Constraint Programming (CP) pode ser usada para resolver o NSP:
  - CP é uma estratégia interessante para resolver problemas fortemente restritos;
  - A exploração da vizinhança pode ser modelada como um problema que é resolvido por técnicas CP.

Explorando Grandes Vizinhanças

- **Discussão:**
- No geral, grandes vizinhanças não podem ser exploradas até otimalidade em tempo polinomial;
- Entretanto, frequentemente NSP ou PNSP podem ser eficientemente resolvidos por métodos exatos;
- Métodos exatos são, em vários casos, bastante rápidos se a dimensão do problema não é muito alta;

Explorando Grandes Vizinhanças

- **Discussão:**
- Os subproblemas gerados podem ser tratados também por métodos aproximativos;
- A literatura ainda carece de estudos que indiquem quais técnicas (heurísticas ou exatas) são preferidas para a exploração de grandes vizinhanças.

Aperfeiçoando Metaheurísticas

- *Métodos exatos* podem ser usados para implementar sub-processos em MHs, tais como:
  - intensificação;
  - diversificação / perturbação.

Aperfeiçoando Metaheurísticas

- **Exemplo: Perturbação no ILS**

Algorithm 4.3.1 *Iterated Local Search*

Step 1: Let s be an initial solution.
Step 2: **while** `stopping_criterion` is not met **do**
(i) Let s' be the solution obtained from s after a perturbation.
(ii) Call `local_search`(s') to produce the solution s''.
(iii) **if** s'' is accepted as the new incumbent solution **then** $s = s''$.
enddo
Step 3: Return s.

Aperfeiçoando Metaheurísticas

- **Exemplo: Perturbação no ILS**
  - A perturbação no ILS pode ser obtida aplicando um movimento aleatório em uma vizinhança grande;
  - Entretanto, desempenhos estado-da-arte são usualmente obtidos via perturbações mais específicas (dependentes do problema);
  - Uma alternativa é determinar a perturbação com um método exato (dificilmente revertida via LS);

Aperfeiçoando Metaheurísticas

- **Exemplo: Perturbação no ILS**
  - Essa ideia pode ser implementada fixando uma parte da solução e deixando o restante livre;
  - A parte livre (subproblema) é otimizada e então reunida à parte fixa;
  - Pode ser necessário tratar restrições violadas.

Aperfeiçoando Metaheurísticas

- **Exemplo: Perturbação no ILS**
  - pseudocódigo da função de perturbação:

begin

Let $s'' = s \setminus r$, where $r \subset s$.
Define $\mathcal{P}(r)$, a subproblem that depends on r.
Let $opt(\mathcal{P}(r))$ be the optimal solution of $\mathcal{P}(r)$.
Let $\bar{s'} = s'' \cup opt(\mathcal{P}(r))$.
Modify $\bar{s'}$ such that feasibility is restored.
Return s', be the modified feasible solution.

end

Aperfeiçoando Metaheurísticas

- **Exemplo: Perturbação no ILS**
  - Lourenço []: ILS aplicado ao *job-shop scheduling problem* (JSP);
  - Ignora as restrições de precedência de duas máquinas selecionadas aleatoriamente;
  - Resolve o problema de scheduling de cada uma dessas máquinas isoladamente (dois subproblemas);
  - Pode ser necessário tratar restrições de precedência para se obter uma solução viável;
  - Por fim, tem-se uma nova solução inicial para a busca local.

Aperfeiçoando Metaheurísticas

- **Outras Estratégias:**
- Emprego de métodos exatos no operador de recombinação de Algoritmos Genéticos
  - visa determinar a melhor solução do subproblema *Recombination(s, s');*
  - subproblema definido a partir da fixação dos elementos comuns das soluções (*s* e *s'*) e variação dos demais;
  - subproblema definido por todos os elementos de *s* e *s'*.

Usando Técnicas *B&B* em Heurísticas de Busca Construtivas

- *Heurísticas construtivas* geram soluções a partir de uma solução parcial inicialmente vazia;
- *Componentes* são adicionados à solução parcial iterativamente até obter uma solução completa;
- Os componentes são adicionados de acordo com uma *função gulosa* (de forma probabilística), considerando-se também alguma *informação heurística*.

Usando Técnicas *B&B* em Heurísticas de Busca Construtivas

- Nas heurísticas construtivas, soluções parciais são usualmente somente *estendidas*, e nunca *reduzidas*;
- Entretanto, essas heurísticas podem ser facilmente transformadas em *algoritmos de busca em árvore*;
- Basta, por exemplo, adicionar um *mecanismo de retrocesso* e considerar *limitantes inferior/superior*;
- Dessa forma, pode-se aplicar B&B!

Usando Técnicas *B&B* em Heurísticas de Busca Construtivas

- **Exemplo: Approximate Nondeterministic Tree Search (ANTS)**
- Aplicado inicialmente à solução do *quadratic assignment problem* (QAP);
- Testa a adição de uma componente à solução parcial corrente e estima seu custo de conclusão via um limitante inferior;
- Quanto menor essa estimativa, mais atrativa é essa componente
- Esta informação heurística é usada para influenciar a probabilidade das decisões ao longo da construção da solução.

Explorando a Estrutura de Boas Soluções

- Em muitos problemas de otimização, as soluções de alta qualidade (*ótimos locais*) possuem um grande número de *componentes em comum*;
- Estes componentes podem ser usados para definir *subproblemas muito promissores*;
- Frequentemente, estes subproblemas são pequenos o suficiente para serem resolvidos por *métodos exatos*.

Explorando a Estrutura de Boas Soluções

- Esta estratégia se divide em duas etapas:
  - Primeiro, um algoritmo aproximativo é usado (repetidamente) para obter um conjunto de *soluções de alta qualidade*;
  - Segundo, um subproblema é definido baseado nos *componentes promissores* pertencentes às soluções de alta qualidade obtidas na etapa anterior.

Explorando a Estrutura de Boas Soluções

- No geral, o subproblema conterá todos os componentes, ou pelo menos os mais importantes, contidos nas soluções obtidas;
- Espera-se que o subproblema possa ser resolvido de forma relativamente simples por um *método exato;*
- A solução ótima do subproblema fornece um *limite superior* para a solução ótima do problema original.

Explorando a Estrutura de Boas Soluções

Algorithm 4.5.1 *Exploiting structure by collecting information*

Step 1: *Initialization*

Let $\mathcal{I} = \emptyset$, where $\mathcal{I}$ is the set of collected solutions.

Step 2: *Approximate algorithm*

while `stopping_criterion` is not met **do**

Let s be the solution returned after running an approximate algorithm.

Add s to $\mathcal{I}$.

enddo

Reduce $\mathcal{I}$ according to some criteria (optional).

Step 3: *Optimization*

Define a subproblem $\mathcal{P}(\mathcal{I})$ that depends on $\mathcal{I}$.

Find $opt(\mathcal{P}(\mathcal{I}))$, the optimal solution of $\mathcal{P}(\mathcal{I})$.

Step 4: Return $opt(\mathcal{P}(\mathcal{I}))$.

Explorando a Estrutura de Boas Soluções

- **Exemplo:** ***Tour Merging (TSP)***
- Inicialmente, gera-se um conjunto de soluções ótimas para o TSP (armazenadas em *I*):
  - *Applegate et al.* [] —> *Lin-Kernighan* (LK);
  - *Cook & Seymour* [] —> *Helsgaun's LK.*
- Posteriormente, resolve-se o TSP sobre um subgrafo restrito definido a partir de *I*:
  - $G' = (V, A')$, $A' = \{a \in A : existe\ t \in I, a \in t\}$

Explorando a Estrutura de Boas Soluções

- Várias estratégias podem ser aplicadas para resolver o TSP no subgrafo reduzido *G′*:
  - método exato (e.g., *Concorde package*);
    - como *G′* é esparso, pode-se usar *branch-width*;
    - *dynamic programming algorithm*;
- Essa abordagem representa uma das mais eficientes para resolver entre instâncias médias e grandes do TSP (2000 a 20.000 cidades).

Explorando a Estrutura de Boas Soluções

- **Discussão**:
- Note que a ideia discutida nessa abordagem não é necessariamente restrita à aplicação de métodos exatos na solução do subproblema;
- Vários trabalhos aplicam métodos SLS ao subproblema reduzido, alcançando excelentes resultados.

Refinamento de Informações de Relaxações em MHs

- Um forma mais usual de combinar elementos de algoritmos exatos e MHs é através do refinamento de informações obtidas de uma relaxação do problema original;
- Essas informações podem ser usadas para “guiar" algoritmos SLS.

Refinamento de Informações de Relaxações em MHs

- **Exemplo: *Simplex* e *Tabu Search***
- *Vasquez & Hao* [] combinam TS e Simplex para tratarem o *0-1 problema da mochila multidimensional:*

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \\ & x_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, n, \end{aligned}$$

Refinamento de Informações de Relaxações em MHs

- Inicialmente, a abordagem relaxa as restrições do problema original;
- Como a solução do problema relaxado pode estar muito distante da solução ótima (inteira), uma restrição adicional é considerada no problema relaxado:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} p_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \\
& \sum_{i=1}^{n} x_i = k \\
& x_i \in [0, 1], \qquad \forall i = 1, \ldots, n
\end{aligned}
$$

Refinamento de Informações de Relaxações em MHs

- Nessa formulação, tem-se $0 \leq k \leq n$. Assim, obtém-se $n+1$ LP problemas (um para cada k).
- Cada subproblema é denotado $P(k)$.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} p_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} r_{ji} x_i \leq R_j, \quad \forall j = 1, \ldots, m \\
& \sum_{i=1}^{n} x_i = k \\
& x_i \in [0, 1], \qquad \forall i = 1, \ldots, n
\end{aligned}
$$

Refinamento de Informações de Relaxações em MHs

- As soluções de *P(k)* podem ser fracionais;
- Entretanto, espera-se que a solução do problema original estará próxima de uma das soluções de *P(k)*;
- *Vasquez & Hao* calculam limitantes para *k*, com o intuito de reduzir o número de subproblemas;
- O Simplex é usado para resolver os problemas *P(k)*;
- As soluções retornadas são usadas para gerar pontos iniciais para uma Busca Tabu.

Refinamento de Informações de Relaxações em MHs

- **Discussão:**
- Refinamento de *limitantes inferiores* obtidos a partir de relaxações Lagrangianas tem se mostrado uma área ativa;
- Vários resultados estado-da-arte para inúmeros COPs foram obtidas dessa forma;

Conclusões

- Existem muitas ***oportunidades de pesquisa*** para o desenvolvimento de algoritmos que integram *busca local* e *métodos exatos;*
- Apesar das vantagens da complementaridade dessas estratégias, relativamente poucos pesquisadores desenvolvem pesquisa nessa área:
  - estes métodos são mais complexos e requerem mais tempo para seu desenvolvimento; e
  - requerem um sólido conhecimento das duas áreas.

EXEMPLO

Coordenação de Relés

- Dois dispositivos de proteção dispostos em série são coordenados somente se, diante de uma falta, o relé mais próximo da falta (relé primário) opera antes do secundário (relé de retaguarda).
- O relé secundário deve operar somente se o primário falhar.
- O tempo de operação de um relé próximo de uma falta é:

$$T_i = TMS_i \left[\alpha_i + \frac{\beta_i}{\left(I_i^{sc} / (MC_i \cdot RCT_i) \right)^{\gamma_i} - 1} \right]$$

Coordenação de Relés

$$T_i = TMS_i \left[\alpha_i + \frac{\beta_i}{\left(I_i^{sc} / (MC_i \cdot RCT_i) \right)^{\gamma_i} - 1} \right]$$

- em que α_i, β_i e γ_i são parâmetros constantes;
- I_i^{sc} é a corrente de curto-circuito vista pelo relé i;
- RCT_i é um parâmetro conhecido;
- TMS_i e MC_i são as variáveis do problema;
- Esta função é claramente não-linear!

Coordenação de Relés

- A coordenação ótima de relés é frequentemente tratada da seguinte forma:

$$\min f_t = \sum_{i=1}^{n} T_i \qquad (CR1)$$

$$T_{ij} - Ti \geq CTI, \quad \forall i \in 1, ..., n, \quad \forall j \in \mathcal{B}_i \qquad (CR2)$$

$$TMS_i \in \mathcal{T}_i, \quad \forall i \in 1, ..., n \qquad (CR3)$$

$$MC_i \in \mathcal{M}_i, \quad \forall i \in 1, ..., n \qquad (CR4)$$

$$T_{ij} = TMS_j \left[\alpha_j + \frac{\beta_j}{\left(I_{ij}^{sc}/(MC_j \cdot RCT_j)\right)^{\gamma_j} - 1} \right] \qquad (CR5)$$

Coordenação de Relés

- (CR1): minimiza tempo de operação dos relés;
- (CR2): restringe tempo de atuação dos relés secundários;
- (CR3): define valores aceitáveis para *TMS*;
- (CR4): define valores aceitáveis para *MC*;
- (CR5) : T_{ij} é o tempo de operação do relé j quanto opera como retaguarda do relé i;
- Os conjuntos $\mathcal{T}_i$ e $\mathcal{M}_i$ podem ser contínuos ou discretos, dependendo do tipo de relé.

Coordenação de Relés

- Algoritmo DE/LP proposto (*Costa et al., 2016*):
  - Método DE evolui valores de *MC*;
  - Modelo LP, chamado pelo DE, determina valores *ótimos* de *TSM* considerando os valores de *MC* fornecidos pelo DE;

Referências

- V. Maniezzo, T. Stutzle, S. Vob (eds.), *Matheuristics: hybridizing metaheuristics and mathematical programming*, Springer, 1st ed., 2009.

- M.H. Costa, R.R. Saldanha, M.G. Ravetti, E.G. Carrano, *Robust coordination of directional overcurrent relays using a matheuristic algorithm*, IET Generation, Transmission & Distribution, pp. 1-11, 2016.