Bacharelado em Ciência da Computação
Métodos Heurísticos

Prof. Diego Mello da Silva

Instituto Federal de Minas Gerais - Campus Formiga

12 de novembro de 2014

Sumário

Fundamentos

Problemas

- **Problema**: Questão geral a ser respondida, geralmente possuindo diversos **parâmetros** (ou variáveis livres) cujos valores são deixados
- Cada problema é dado por:
  - Uma descrição geral de todos seus parâmetros
  - Uma declaração de que propriedades a **solução** é requerida satisfazer
- Uma **instância** do problema é obtida especificando valores particulares para todos os parâmetros do problemas
- **Exemplo**: Problema do Caixeiro Viajante
  - Parâmetro 1: conjunto finito $C = \{c_1, c_2, \ldots, c_m\}$ de cidades
  - Parâmetro 2: distância $d\left(c_i, c_j\right)$ entre pares de cidades $c_i, c_j \in C$
  - Solução: ordenação de cidades $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)} \rangle$ que minimiza

$$\left(\sum_{i=1}^{m-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) \right) + d\left(c_{\pi(m)}, c_{\pi(1)}\right)$$

  - Instância: $C = \{c_1, c_2, c_3, c_4\}$, $d(c_1, c_2) = 10$, $d(c_1, c_3) = 5$, $d(c_1, c_4) = 9$, $d(c_2, c_3) = 6$, $d(c_2, c_4) = 9$, $d(c_3, c_4) = 3$.

Problemas de Otimização Combinatória

- São expressos em termos de encontrar o **melhor** conjunto possível de valores para um grupo de **variáveis**

- **Exemplo 1**: **Problema da Mochila 0-1** (*Knapsack Problem*)

- Problema: atribuir 0 ou 1 em x_j, com $1 \leq j \leq n$ tal que maximize o valor total $\sum c_j x_j$ respeitando a capacidade K da mochila

- Modelagem Matemática

$$
\begin{aligned}
\text{max} \quad & \sum_{j=1}^{n} c_j x_j & \text{(Maximizar Valor)} \\
\text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j \leq K & \text{(Restrição de Capacidade)} \\
& x_j \in \{0, 1\}
\end{aligned}
$$

- Testar todas as combinações de 0 e 1 nas n variáveis: $O\left(2^n\right)$

- Muitos problemas são da classe $\mathcal{NP}$ e, portanto, são resolvidos por **busca local**

Problemas de Otimização Combinatória

- **Exemplo 2**: **Caixeiro Viajante** (*Traveling Salesman Problem*) (Modelo 1)

- Problema: atribuir 0 ou 1 em x_e, com $x_e \in E$ tal que minimize o custo total do ciclo que passa por todos os nós $v \in V$ do grafo $G = (V, E)$

- Modelagem Matemática I

$$
\begin{array}{llll}
\min & \displaystyle\sum_{e \in E} x_e c_e & & \text{(Minimizar Custo do Ciclo)} \\
\text{s.t.} & \displaystyle\sum_{e \in \delta(v)} x_e = 2 & v \in V & \text{(Cada nó } v \text{ tem grau 2)} \\
& \displaystyle\sum_{e \in E(S)} x_e \leq |S| - 1 & S \subset V & \text{(Eliminação de Subciclos)} \\
& x_e \in \{0, 1\} & e \in E &
\end{array}
$$

- Ciclo: subgrafo conexo onde cada nó tem grau 2

- $E(S)$: conjunto dos arcos e formados entre pares de nós (u, v) tal que $u, v \in S$

Problemas de Otimização Combinatória

- **Exemplo 2**: **Caixeiro Viajante** (*Traveling Salesman Problem*) (Modelo 2)
- Problema: atribuir 0 ou 1 em x_{ij}, com $(i, j) \in E$ tal que minimize o custo total do ciclo que passa por todos os nós $v \in V$ do grafo $G = (V, E)$
- Modelagem Matemática II

$$
\begin{array}{llll}
\min & \displaystyle\sum_{i=1}^{n}\sum_{i=j}^{n} x_{ij}c_{ij} & c_{ij} = \infty \text{ para todo } i = j & \text{(Minimizar Custo do Ciclo)} \\
\text{s.t.} & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & j = 1, 2, \ldots, n & \text{(Em cada nó } j \text{ chega 1 arco)} \\
 & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & i = 1, 2, \ldots, n & \text{(De cada nó } i \text{ parte 1 arco)} \\
 & x_{ij} \in \{0, 1\} & &
\end{array}
$$

- Ciclo: subgrafo conexo onde cada nó tem grau 2
- Variável de Decisão: $x_{ij} = \begin{cases} 1, \text{se nó } j \text{ é alcançado a partir do nó } i \\ 0, \text{caso contrário} \end{cases}$

Complexidade

Motivação (Adaptado de [Garey e Johnson])

"I can't find an efficient algorithm, but neither can all these famous people."

Algoritmos Polinomiais e Problemas Intratáveis

- Embora algoritmos possuam diferentes funções de complexidade de tempo, o que é 'eficiente' depende da situação em questão
- No entanto, cientistas da computação reconhecem claramente uma distinção simples que ofecere *insights* no assunto
- É a distinção entre **algoritmos de tempo polinomial** e **algoritmos de tempo exponencial**
- Uma função $f(n)$ é $O\big(g(n)\big)$ se existe uma constante c tal que

$$\boxed{\big|f(n)\big| \leq c \cdot \big|g(n)\big|, \forall n \geq 0}$$

- **Algoritmo de Tempo Polinomial**: é aquele cuja função de complexidade é $O\big(p(n)\big)$, para alguma função polinomial p com tamanho de entrada n
- **Algoritmos de Tempo Exponencial**: algoritmos cuja função de complexidade de tempo não pode ser limitada
- A distinção entre estes dois tipos de algoritmos é central na nossa noção de **intratabilidade** e teoria de **NP-Completude**

Problemas $\mathcal{P}$ e $\mathcal{NP}$

- Problemas $\mathcal{N}$: problemas que podem ser resolvidos por **algoritmos determinísticos** em **tempo polinomial**
- Problemas $\mathcal{NP}$: problemas para os quais não é conhecido existir algoritmos determinísticos polinomiais, mas que podem ser resolvidos por **algoritmos não-determinísticos** em **tempo polinomial**
- Problemas $\mathcal{NP}$ são computacionalmente relacionados
- Um problema $\mathcal{NP}$ pode ser resolvido em tempo polinomial se e somente se todos os outros problemas em $\mathcal{NP}$ também puderem
- Problema de Decisão **Sim/Não**
- TSP versão de decisão:
  - Seja constante k, e $C = \{c_1, c_2, \ldots, c_n\}$ e distância $d(c_i, c_j)$
  - Existe roteiro para cidades em C com comprimento total $\leq k$?
- Sobre Problema de Decisão:
  - Solução pode ser **verificada** facilmente
  - Solução pode ser muito difícil/impossível de ser obtida, mas uma vez conhecida pode ser verificada facilmente

Algoritmos Não-Determinísticos

- **Alg. Não-Determinístico**: escolhe uma dentre várias alternativas possíveis a cada passo
- Função **`escolhe(`***C***`)`**:
  - Escolhe um dos elementos de C de forma arbitrária, em tempo $O(1)$
  - Se conjunto de possibilidades leva à resposta, este conjunto é escolhido
- **Máquina Não-Determinística**: produz cópias de sí mesma quando diante de duas ou mais alternativas, computando-as independentemente.
- Versão Não-Determinística: Busca $\left(O(1) \text{ vs. } O(n)\right)$ e Ordenação $\left(O(n) \text{ vs. } O(n \log n)\right)$

Algoritmo 1 Search(A, 1, n)

```
j ← escolhe(A, 1, n)
if (A[j] = x) then
    return SUCESSO
else
    return INSUCESSO
end if
```

Algoritmo 2 Sort(A, 1, n)

```
for (i ← i to n) do
    B[i] ← 0
end for
for (i ← 1 to n) do
    j ← escolhe(A, 1, n)
    if (B[j] = 0) then
        B[j] ← A[i]
    else
        return INSUCESSO
    end if
end for
```

Problema da Satisfabilidade (**SAT**)

- Seja um conjunto de variáveis booleanas $x_1, x_2, \ldots, x_n$
- **Forma Normal Conjuntiva**: expressão booleana formada por conjunções de cláusulas, cada qual formada por disjunção de variáveis booleanas
- Exemplos:

  $E_1 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee x_2) \wedge (x_3)$

  $E_2 = (x_1 \vee x_4) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3) \wedge (\neg x_4)$

  $E_3 = (\neg x_1) \wedge (x_2 \vee x_3)$
- **Problema da Satisfabilidade (Cook, 1970)**: dada uma expressão na FNC, existe uma atribuição de valores lógico verdadeiro/falso às variáveis x_i, $i = 1, \ldots, n$ que torne a expressão verdadeira?
- $E_1 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee x_2) \wedge (x_3)$ é satisfatível com $x_1 = F$, $x_2 = V$, $x_3 = V$
- $E_4 = (x_1) \wedge (\neg x_1)$ não é satisfatível
- Melhor algoritmo determinístico: $O\left(2^n\right)$
- **CIRCUIT-SAT** foi o primeiro problema $\mathcal{NP}$-Completo (ver Lema 34.5 e 34.6 de [Cormen et al])
- **BOOLEAN-SAT** é também $\mathcal{NP}$-Completo

Problema da Satisfabilidade (**SAT**) é $\mathcal{NP}$

- Para mostrar que um problema é $\mathcal{NP}$, basta apresentar um algoritmo não-determinístico polinomial para resolvê-lo, ou apresentar um algoritmo determístico polinomial para verificá-lo (i.e, um **certificado**)

- **Algoritmo Não-Determinístico**: obtém uma das 2^n atribuições possível em tempo $O(n)$

Algoritmo 3 SAT-Ndet(E, n)

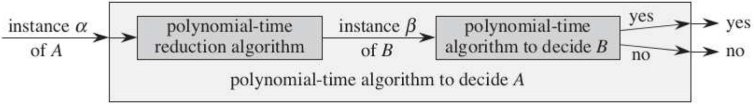
```
for (i ← 1 to n) do
    x_i ← escolhe(TRUE, FALSE)
    if (E(x_1, x_2, ..., x_n) = TRUE) then
        return SUCESSO
    else
        return INSUCESSO
    end if
end for
```

- **Certificado**

  Verificar se uma atribuição de valores lógicos para as x_i, $i = 1, \ldots, n$ variáveis booleanas faz a expressão na FNC satisfazível

Problemas $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-Difícil e $\mathcal{NP}$-Completo

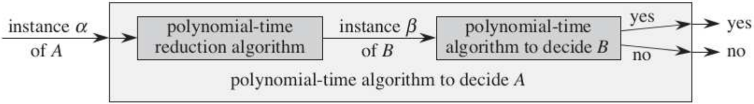
- Problemas $\mathcal{P}$: algoritmos determinísticos **em tempo polinomial**
- Problemas $\mathcal{NP}$: algorítmos não-determísticos **em tempo polinomial**
- $\mathcal{P} \subseteq \mathcal{NP}$: algoritmos determinísticos são caso especial de não-determinísticos
- Problema mais famoso em Computação: $\mathcal{P} = \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$?
- Redução polinomial



- Problemas são **polinomialmente equivalentes** se $\Pi_1 \propto \Pi_2$ e $\Pi_2 \propto \Pi_1$
- Problemas $\mathcal{NP}$-Difícil: **SAT** $\propto \Pi$
- Problemas $\mathcal{NP}$**-Completo**: $\mathcal{NP}$ + $\mathcal{NP}$-Difícil
  - $\Pi \in \mathcal{NP}$
  - $\forall\ \Pi' \in \mathcal{NP}$-Completo, $\Pi' \propto \Pi$
- Problemas $\mathcal{NP}$-Difícil são **tão difíceis de resolver quanto** $\mathcal{NP}$-Completo

Redução de Problemas

- Como mostrar que um problema Π é $\mathcal{NP}$-Completo
  (1) Mostre que o problema Π é $\mathcal{NP}$
  (2) Mostre que um problema $\mathcal{NP}$-Completo conhecido pode ser reduzido à Π
- Redução:



- É possível porque Cook mostrou que *SAT* é $\mathcal{NP}$-Completo
- Redução polinomial é transitiva: $\boxed{\textbf{SAT} \propto \Pi_1 \wedge \Pi_1 \propto \Pi_2 \rightarrow \textbf{SAT} \propto \Pi_2}$
- **Exemplo**: Mostrando que TSP é $\mathcal{NP}$-Completo

Passo 1: TSP é $\mathcal{NP}$

- **Opção 1**: Apresentar um algoritmo não-determinístico polinomial para o TSP

Algoritmo 4 TSP-NDet(G)

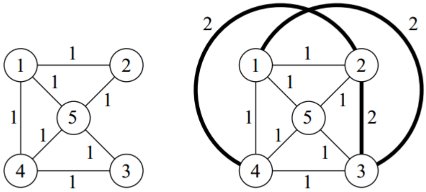
1: $i \leftarrow 1$
2: **for** ($t \leftarrow 1$ **to** $G.V$) **do**
3: $j \leftarrow$ **`escolhe(`**i**`, Adj[`**i**`])`**
4: **`antecessor[`**j**`]`** $\leftarrow i$
5: **end for**

- **Opção 2**: Solução do TSP pode ser **verificada** em tempo polinomial

  Certificado: verificar que uma solução do TSP é uma ordenação das m cidades contidas em C

Passo 2: Ciclo Hamiltoniano $\propto$ TSP

- Ciclo hamiltoniano é conhecido ser $\mathcal{NP}$-Completo
- Reduzir, em tempo polinomial, **ciclo hamiltoniano** em **TSP**
- Redução: dado instância G_1 do ciclo hamiltoniano, construa G_2 do TSP
  (1) Para cidades, use os vértices de G_1
  (2) Para distâncias, use 1 se existir arco em G_1, e 2 se não existir arco em G_1
- Use o TSP para encontrar um roteiro menor ou igual a V em G_2
- Roteiro encontrado é um ciclo hamiltoniano em G_1
- Exemplo:



Métodos Heurísticos

Heurísticas *versus* Metaheurísticas

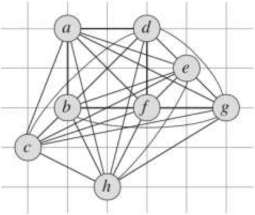
- **Heurísticas**: conjunto de passos bem definidos para **identificar rapidamente** soluções de alta qualidade em um **problema específico**.
- Exemplos de Heurísticas para o TSP
  - Vizinho mais Próximo
  - Cristophides
  - Árvore Geradora Mínima
- **Busca Local**: funcionam partindo de uma configuração inicial realizando pequenas mudanças nesta configuração até que o o estado atingido seja o melhor possível
- **Metaheurísticas**: estratégias gerais alto-nível que guiam a busca por boas soluções. Logo, são usadas para realizar busca local.
- Exemplos de metaheurísticas
  - Algoritmos Genéticos
  - Simulated Annealing (Recozimento/Têmpera Simulada)
  - Iterated Local Search
  - GRASP

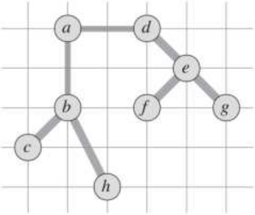
TSP: Algoritmo da Árvore Geradora Mínima

Algoritmo 5 TSP-MST(G)

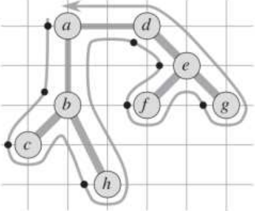
1: Selecionar um nó r de G para ser a raíz da MST
2: Computar a árvore geradora mínima T usando Prim
3: Seja H a lista de nós ordenados segundo a visitação pré-ordem de T
4: **return** (H)

TSP: Algoritmo da Árvore Geradora Mínima

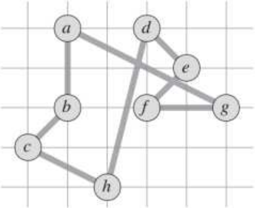




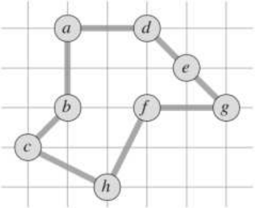
Árvore gerado mínima



Percuso pré-ordem



Caminho encontrado (custo: 19,074)



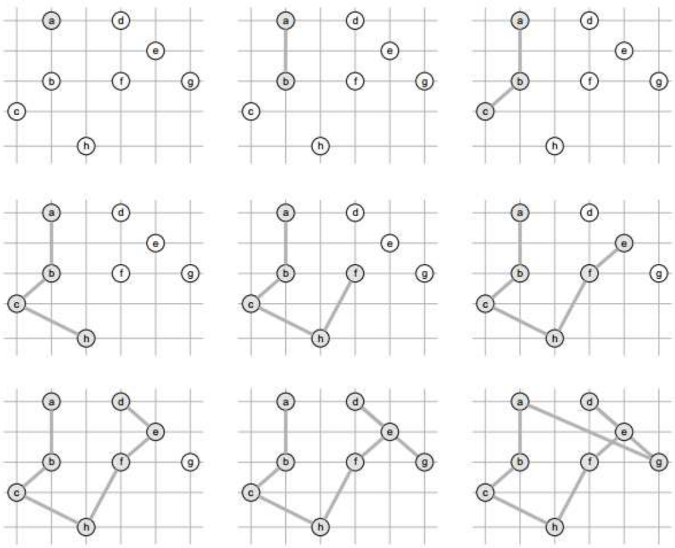
Menor caminho (custo: 14,715)

TSP: Algoritmo do Vizinho Mais Próximo

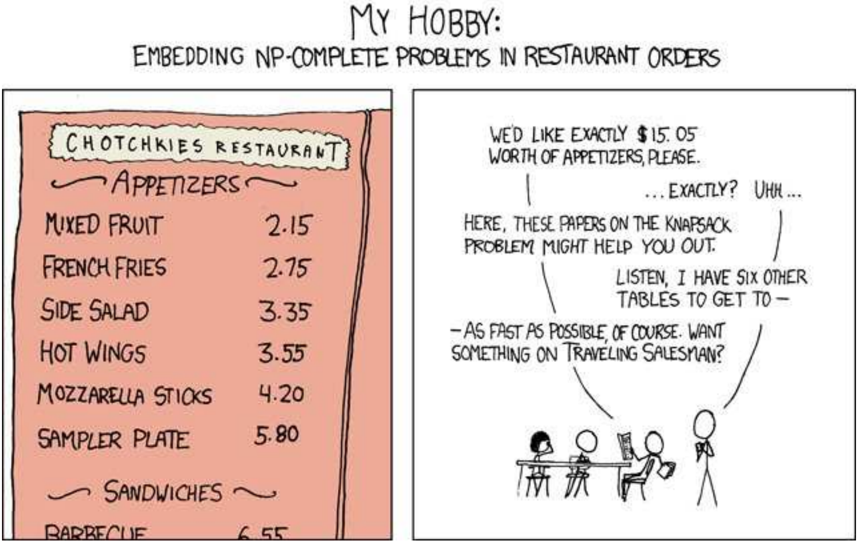
Algoritmo 6 TSP-MST(G)

Selecionar um nó v_0 inicial de G
$C \leftarrow \varnothing$
Add(C, v_0)
$v \leftarrow v_0$
while ($C < |G.V|$) **do**
    $u \leftarrow$ vértice mais próximo de v, $u \in Adj[v]$, $u \notin C$
    Add(C, u)
    $v \leftarrow u$
end while
Add(C, v_0)
return (C)

TSP: Algoritmo do Vizinho Mais Próximo



Caminho final: a, b, c, h, f, e, d, g, a
Custo: 16,5


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO —
— AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Benchmarks

Alguns Benchmarks para Problemas de Otimização

- TSPLIB: Travelling Salesman Problem
  `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`
- PSPLIB: Project Scheduling Problem
  `http://www.om-db.wi.tum.de/psplib/main.html`
- MPLIB: Mix Integer Problem
  `http://miplib.zib.de/miplib2010.php`
- OR-Library: Operations Research Library
  `http://www.brunel.ac.uk/~mastjjb/jeb/info.html`
- SATLIB: Satisfiability Problem
  `http://www.cs.ubc.ca/~hoos/SATLIB/`
- Clique Benchmark: Clique Problem
  `http://www.cs.hbg.psu.edu/txn131/clique.html`
- Global Optimization Test Problems
  `http://www.mat.univie.ac.at/~neum/glopt/test.html`
- DIMACS: Center for Discrete Mathematics and Theoretical Computer Science
  `http://dimacs.rutgers.edu/`

Outros Problemas Combinatórios

Alguns Problemas Combinatórios Clássicos

- Problema da Clique Máxima
- Problema da Árvore Geradora com Restrição de Grau
- Problema Subset-Sum
- Problemas Set Covering, Set Packing e Set Partition
- Problema do Conjunto Independente (ou Estável)
- Problema da Cobertura de Vértices
- Problema da Coloração de Vértices
- Problema da Mochila Generalizada
- Problema da Árvore Geradora Mínima Capacitada
- Problema Job-Shop Scheduling

Problema da Clique Máxima

- Uma **clique** de um grafo G é um subgrafo H de G onde para cada par de vértices u e v em H, então $(u, v) \in E(H)$.
- Uma clique é de tamanho k se existem k vértices no clique.
- Definimos como $\omega(G)$ como o tamanho da **clique máxima** no grafo G, tal que não exista nenhum clique de tamanho $\omega(G) + 1$ ou maior em G
- Logo, clique máxima é a **clique com maior cardinalidade**

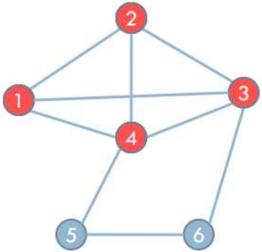
- Formulação de Arcos

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} x_i \\ \text{s.t.} \quad & x_i + x_j \leq 1, \quad \forall (i,j) \in \overline{E} \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n \end{aligned}$$

onde

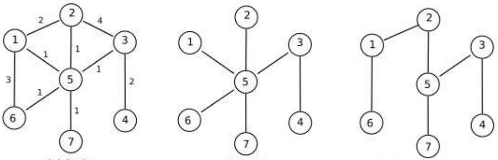
$$\overline{E} = \left\{ (i,j) | i,j \in V, i \neq j, (i,j) \notin E \right\}$$

- Exemplo:



Problema da Árvore Geradora com Restrição de Grau

- Dado um grafo valorado $G = (V, E)$ e inteiros positivos $b_1, b_2, \ldots, b_n$, encontrar uma árvore geradora de custo mínimo T tal que o grau de cada vértice em T é no máximo b_i
- Exemplo:



(a) Rede (b) MST (c) DCMST (b_i=3)

- Formulação, de Narula e Ho (1980)

$$
\begin{array}{llll}
\min & \displaystyle\sum_{i,j \in V, i \neq j} c_{ij} x_{ij} & & \\
\text{s.t.} & \displaystyle\sum_{j \in V, i \neq j} x_{ij} \leq b_i & \forall\, i \in V & \text{(Limite Grau)} \\
& \displaystyle\sum_{j \in V, i \neq j} x_{ij} \geq 1 & \forall\, i \in V & \text{(Conectividade)} \\
& \displaystyle\sum_{i,j \in S} x_{ij} \leq |S| - 1 & \forall\, S \subseteq V & \text{(Elim. Subciclos)} \\
& x_{ij} \in \{0, 1\} & &
\end{array}
$$

- Para n vértices, teremos:
  - (a) $\dfrac{n(n-1)}{2}$ variáveis
  - (b) $2^n + (n-1)$ restrições
- Problema cresce rapidamente mesmo para tamanho moderado

Problema da Árvore Geradora com Restrição de Grau

- Heurística **Primal** de Narula e Ho (1982)
  (1) Calcula uma solução factível para o **Degree-Constrained Spanning Tree** usando uma versão modificada do Prim onde a inclusão da aresta na árvore não pode violar restrição de grau imposta a cada nó de G
  (2) Varre cada arco e_{ij} de T e verifica se:
  (a) Arco pode ser substituído por outro de menor peso e_{uv}
  (b) Arco pode ser substituído por arco e_{uv} de peso igual À e_{ij} tal que os nós u, v ou ambos não excedam o grau máximo permitido e os nós i, j ou ambos já estão no limite do grau máximo.

- Heurística **Dual** de Narula e Ho (1982)
  (1) Calcula solução do MST, que é limite inferior do DCMST
  (2) Considere nó i tal que $d_i > b_i$. Para cada arco e_{ij}, encontrar arco e_{rs} de substituição admissível tal que:
  (a) Remoção de e_{ij} e inclusão de e_{rs} resulte em árvore
  (b) Restrições de grau não sejam violadas para r, s ou ambos
  (c) Penalidade $[custo(e_{rs}) - custo(e_{ij})]$ é mínimo dentre todos os arcos e_{rs}
  (3) Realize a troca de arcos e_{ij} por e_{rs} até que $d_i < b_i$ para todo i

Problema Subset-Sum

- Sejam números $w_1, w_2, \ldots, w_n$, e $X \subset \{1, 2, \ldots, n\}$. Seja $p(X) = \sum_{i \in X} w_i$
- **Subset-Sum**: Dados números naturais $w_1, w_2, \ldots, w_n$, e constante c, existe um sub-conjunto X de $\{1, 2, \ldots, n\}$ tal que $p(X) = c$?
- Exemplo: Para instância $(30, 40, 10, 15, 10, 60, 54)$ e $c = 55$, temos $X = \{2, 4\}$ e $X = \{1, 4, 5\}$
- **Subset-Sum Generalizado**: Dados números naturais $w_1, w_2, \ldots, w_n$ e constante c, encontrar um sub-conjunto X de $\{1, 2, \ldots, n\}$ que maximize $p(X)$ sujeito a restrição de que $p(x) \leq c$

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} w_j x_j \\ s.t. \quad & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n \end{aligned}$$

- Mundo real: armazém com grande quantidade de caixas de pesos variados. Caminhão com capacidade c disponível para transportar caixas. Qual a maior carga possível para embarcar?

Problema Set Covering

- Seja $M = \{1, \ldots, m\}$ e $N = \{1, \ldots, n\}$ conjuntos. Seja $M_1, M_2, \ldots, M_n$ uma dada coleção de sub-conjuntos de M. Cada conjunto M_j na coleção possui peso c_j.
- Dizemos que um sub-conjunto F de N é uma cobertura de M se

$$\boxed{\cup_{j \in F} M_j = M}$$

- O peso de um sub-conjunto F de N é dado por $\sum_{j \in F} c_j$
- Formulação:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ s.t. \quad & \sum_{j=1} a_{ij} x_j \geq 1 \quad i = 1, \ldots, m \\ & x_j \in \{0, 1\} \quad\quad j = 1, \ldots, n \end{aligned}$$

$$\min_F = \left\{ \sum_{j \in F} : F \text{ é um cobertura} \right\} = \min \left\{ \sum_{j=1}^{n} c_j x_j : Ax \geq e, x \in \{0,1\}^n \right\}$$

- A matriz A é uma matriz de incidência, e o vetor **e** é um vetor de 1's

Problema Set Covering: Exemplo

- Um pronto-socorro precisa manter médicos em plantão, de forma que um indivíduo qualificado está disponível para realizar todos os procedimento médico que podem ser necessários.
- Para cada médico disponível em plantão é pago um adicional de salário. Os procedimentos que cada um realiza são conhecidos.

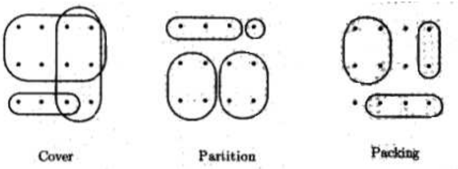
	Médico 1	Médico 2	Médico 3	Médico 4	Médico 5	Médico 6
Procedimento 1	X			X		
Procedimento 2	X				X	
Procedimento 3		X	X			
Procedimento 4	X					X
Procedimento 5		X	X			X
Procedimento 6		X				

- Objetivo: escolher médicos tais que cada procedimento seja coberto com custo mínimo
- Decisão: $x_j = 1$ se médico j está no plantão, 0 caso contrário

$$\begin{array}{llll} \min & \sum_{j=1}^{n} c_j x_j & & \text{(Salário Pago)} \\ s.t. & \sum_{j=1} a_{ij} x_j \geq 1 & i = 1, \ldots, m & \text{(Pelo Menos 1 Médico/Procedimento)} \\ & x_j \in \{0, 1\} & j = 1, \ldots, n & \end{array}$$

Problemas Set Covering, Packing e Partition

- Problemas relacionados: cobertura, empacotamento e particionamento



- **Set Covering**: $F \subseteq \{1, \ldots, n\}$ tal que $\cup_{j \in F} M_j = M$

$$\min_F = \left\{ \sum_{j \in F} : F \text{ é um cobertura} \right\} = \min \left\{ \sum_{j=1}^{n} c_j x_j : Ax \geq e, x \in \{0,1\}^n \right\}$$

- **Set Packing**: $F \subseteq \{1, \ldots, n\}$ tal que $M_j \cap M_k = \varnothing$ para todo $j, k \in F, j \neq k$

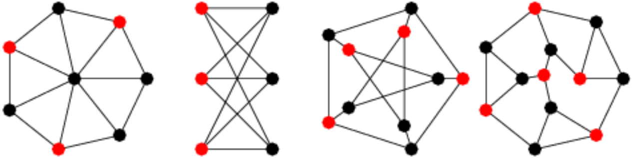
$$\min_F = \left\{ \sum_{j \in F} : F \text{ é um empacotamento} \right\} = \min \left\{ \sum_{j=1}^{n} c_j x_j : Ax \leq e, x \in \{0,1\}^n \right\}$$

- **Set Partition**: $F \subseteq \{1, \ldots, n\}$ que é tanto **cobertura** quanto **empacotamento**

$$\min_F = \left\{ \sum_{j \in F} : F \text{ é um partição} \right\} = \min \left\{ \sum_{j=1}^{n} c_j x_j : Ax = e, x \in \{0,1\}^n \right\}$$

Problema do Conjunto Independente (ou Estável)

- Um **conjunto independente** de um grafo $G = (V, E)$ é um sub-conjunto $S \subset V$ tal que não existem dois nós em S que são adjacentes em G.
- Determinar o maior sub-conjunto S de nós de V que seja um conjunto independente é um problema NP-Difícil, e é o **complemento** do problema da **clique máxima**.
- Exemplos (`http://mathworld.wolfram.com/IndependentSet.html`)



- Formulação:

$$
\begin{aligned}
\max \quad & \sum_{i \in V} x_i \\
s.t. \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E \\
& x_i \in \{0, 1\}
\end{aligned}
$$

Problema do Conjunto Independente (ou Estável)

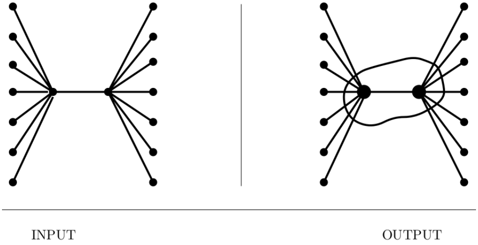
- Algoritmo Guloso:

Algoritmo 7 INDEPENDENT-SET-GREEDY(G)

1: $S \leftarrow \varnothing$
2: **while** ($G \neq \varnothing$) **do**
3: Seja v o nó de menor grau de G
4: $S \leftarrow S \cup \{v\}$
5: Remova v e seus vizinhos de G
6: **end while**
7: **return** S

Problema da Cobertura de Vértices

- Dado um grafo $G = (V, E)$, o problema da **cobertura de vértices** consiste em determinar um sub-conjunto $S \subset V$ tal que cada arco $(u, v) \in E$ incide em pelo menos um nó de S



- Formulação Original (IP)

$$\begin{aligned} \min \quad & \sum_{i \in V} x_i \\ s.t. \quad & x_i + x_j \leq 1 \quad \forall (i, j) \in E \\ & x_i \in \{0, 1\} \end{aligned}$$

- Formulação Relaxada (LP)

$$\begin{aligned} \min \quad & \sum_{i \in V} x_i \\ s.t. \quad & x_i + x_j \leq 1 \quad \forall (i, j) \in E \\ & x_i \geq 0 \\ & -x_i \geq -1 \end{aligned}$$

Problema da Cobertura de Vértices

- Algoritmo de Arredondamento LP[1]

Algoritmo 8 VERTEX-COVER-LP($G = (V, E)$)

```
Construa relaxação LP para uma dada instância G
Invoque um LP solver de tempo polinomial para obter x* ∈ ℝ que minimiza ∑ x_i, i ∈ V
S ← ∅
for (i ← 1 to |V|) do
    if (x_i* ≥ 1/2) then
        S ← S ∪ {i}
    end if
end for
return S
```

[1] http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/nanda-scrib

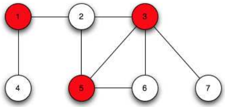
Problema da Cobertura de Vértices

- Algoritmo 2-Aproximação para a Cobertura de Vértices

Algoritmo 9 VERTEX-COVER-2APPROX($G = (V, E)$)

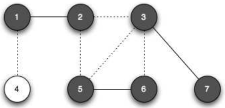
$S \leftarrow \varnothing$
$E' \leftarrow E$
while ($E' \neq \varnothing$) **do**
    Seja (u, v) um arco arbitrário de E'
    $S \leftarrow S \cup \{u, v\}$
    Remova de E' cada arco que incide ou em u ou em v
end while
return S

- Solução Ótima



$S^* = \{1, 3, 5\}$, $|S^*| = 3$

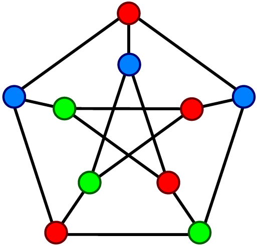
- Solução Aproximada



$S = \{1, 2, 3, 5, 6, 7\}$, $|S| = 2 \cdot |S^*| = 6$

Problema da Coloração de Vértices

- Dado um grafo $G = (V, E)$, colorir os vértices de G usando a menor quantidade de cores tal que para cada arco $(i, j) \in E$, os nós i e j tenham cores diferentes.
- Uma k**-coloração própria** de nós em G é uma atribuição de cores $\{1, 2, \ldots, k\}$ aos vértices de G tal que dois vértices adjacentes não possuam a mesma cor.
- Uma k-coloração **particiona** V em k diferentes **classes de cores**, onde cada membro da classe tem a mesma cor.
- O **número cromático** $\chi(G)$ de G é o menor inteiro k para o qual G é k-colorável



Exemplo: $\chi(G) = 3$

(1): Minimiza o número de cores usadas. (2): Cada nó recebe 1 cor. (3): Cores $\neq$ para nós adjacentes (3): São usadas no máximo k cores. (4): Integralidade de y_k e x_{ik}. Variável $x_{ik} = 1$ se cor k é atribuída a nó i

$$
\begin{array}{llll}
\min & \sum_{k=1}^{n} y_k & & (1) \\
s.t. & \sum_{k=1}^{n} x_{ik} = 1 & \forall\, i \in V & (2) \\
& x_{i,k} + x_{jk} \leq 1 & \forall\, (i,j) \in E & (3) \\
& y_k \geq x_{ik} & \forall\, i \in V, k = 1, \ldots, n & (4) \\
& y_k, x_{ik} \in \{0, 1\} & \forall\, i \in V, k = 1, \ldots, n & (5)
\end{array}
$$

Problema da Coloração de Vértices

- Seja $K = (v_1, v_2, \ldots, v_n)$ uma sequência de vértices de G
- Heurísticas simples para o problema da coloração de vértices:

Algoritmo 10 VERTEX-COLOURING-GREEDY($G = (V, E)$, K)

1: **for** ($i \leftarrow 1$ **to** n) **do**
2: $\quad v \leftarrow i$-ésimo vértice de K
3: $\quad v \leftarrow$ menor índice de cor ‘possível’
4: **end for**

Algoritmo 11 VERTEX-COLOURING-RANDOM-SEQ($G = (V, E)$)

1: $K \leftarrow$ sequência de vértices geradas aleatoriamente
2: **VERTEX-COLOURING-GREEDY(**G, K**)**

Algoritmo 12 VERTEX-COLOURING-LARGEST-FIRST($G = (V, E)$)

1: $K \leftarrow$ sequência de vértices arranjados em ordem decrescente de grau
2: **VERTEX-COLOURING-GREEDY(**G, K**)**

Problema da Mochila Generalizada

- É o problema que generaliza a mochila para m mochilas a serem carregadas, cada qual com capacidade $b_j, j = 1, \ldots, m$
- Existem n itens que precisam ser distribuídos nas m mochilas. Cada item possui peso p_k ($k = 1, \ldots, n$) e utilidade (ou valor) c_k ($k = 1, \ldots, n$).
- Formulação (Goldbarg e Luna):

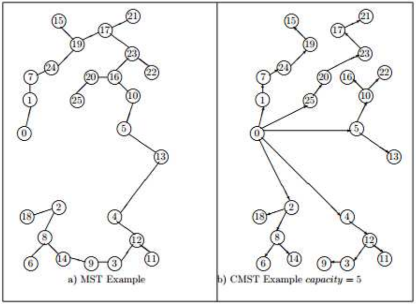
$$
\begin{array}{llll}
\max & \sum_{j=1}^{m} \sum_{i=1}^{n} c_i x_{ij} & & \text{(Maximizar o Valor Carregado)} \\
s.t. & \sum_{i=1}^{n} p_i x_{ij} \leq b_j & \forall\, j \in \{1, \ldots, m\} & \text{(Capacidade Máxima/Mochila)} \\
 & \sum_{j=1}^{m} x_{ij} \leq 1 & \forall\, i \in \{1, \ldots, n\} & \text{(Item } i \text{ Em Apenas 1 Mochila)} \\
 & x_{ij} \in \{0, 1\} & \forall\, i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} & \text{(Integralidade)}
\end{array}
$$

- Variável $x_{ij} = 1$ indica que item i é guardado na mochila j; 0 caso contrário

Problema da Árvore Geradora Mínima Capacitada

- Seja $G = (V, E)$ um grafo não-direcionado e conexo. Cada vértice $v \in V$ possui peso $b_i \geq 0$, com $b_0 = 0$, e representa uma demanda requerida. Cada arco $(i, j) \in E$ possui um custo c_{ij} associado com seu uso na árvore geradora mínima.

- O problema **árvore geradora mínima capacitada** consiste em encontrar uma AGM sobre G centralizada no nó 0 cuja soma dos pesos dos nós de qualquer sub-árvore conectada ao nó 0 não pode ser maior do que capacidade K



a) MST Example

b) CMST Example *capacity* = 5

Seja $b_i = 1, i \in V - \{0\}$, com $K = 5$

- Formulação MIP (Gavish, 1983)

$$\min \quad \sum_{i=0}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$s.t. \quad \sum_{i=0}^{n} x_{ij} = 1 \qquad j = 1..n \tag{2}$$

$$\sum_{i=0}^{n} y_{ij} - \sum_{i=1}^{n} y_{ji} = 1 \qquad j = 1..n \tag{3}$$

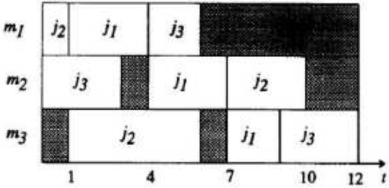
$$x_{ij} \leq y_{ij} \leq (K - b_i) x_{ij} \qquad i = 0..n; j = 1..n \tag{4}$$

$$x_{ij} \in \{0, 1\}, y_{ij} \geq 0 \qquad i = 0..n; j = 1..n \tag{5}$$

Problema Job-Shop Scheduling

- O problema **job-shop scheduling** consiste em **distribuir tarefas** entre máquinas de forma a **reduzir o tempo** necessário para terminar a execução de todas (**makespan**)
- Seja um conjunto de *n* tarefas ou **jobs**, e um conjunto *m* de máquinas.
- Cada tarefa contém um **conjunto de operações** cuja **ordem** de execução é pré-definida.
- Para cada operação é conhecido em que **máquina** ela é executada e o **tempo** necessário para que seja finalizada
- Restrições:
  - (a) Uma tarefa não pode ser atribuída duas vezes na mesma máquina
  - (b) Não existe ordem de execução em relação à operações de tarefas diferentes
  - (c) Uma operação não pode ser interrompida (não-preemptiva)
  - (d) Cada máquina processa apenas uma tarefa por vez
- Exemplo de Instância:

	Operações		
Job	1	2	3
Tempo de Processamento			
j_1	3	3	2
j_2	1	5	3
j_3	3	2	3
Sequencia de máquinas			
j_1	m_1	m_2	m_3
j_2	m_1	m_3	m_2
j_3	m_2	m_1	m_3



Problema Job-Shop Scheduling

- Sejam os parâmetros para o JSSP

  y_{ij}: Tempo inicial da operação j na máquina i

  p_{ij}: duração da tarefa j na máquina i

  N: Conjunto de todas as operações (i,j)

  A: Conjunto de restrições de rotas $(i,j) \to (k,j)$

  m: Número total de máquinas

- Formulação LP

$$
\begin{array}{lllr}
\min & C_{\max} & & (1) \\
s.t. & y_{kj} - y_{ij} \geq p_{ij} & \forall\, (i,j) \to (k,j) \in A & (2) \\
 & C_{\max} - y_{ij} \geq p_{ij} & \forall\, (i,j) \in N & (3) \\
 & y_{ij} - y_{il} \geq p_{il} \textbf{ ou } y_{il} - y_{ij} \geq p_{ij} & \forall\, (i,l) \textbf{ e } (i,j), i = 1, \ldots, m & (4) \\
 & y_{ij} \geq 0 & \forall\, (i,j) \in N & (5)
\end{array}
$$

- Observe que embora formulação seja linear, restrição (4) impede que seja resolvido como problema de programação linear convencional

Outros Problemas Combinatórios Interessantes[2]

- Problema da Árvore de Steiner (STP, TSTP)
- Problema da Coloração de Arcos (ECP)
- Problema do Caminho Hamiltoniano (HPP) e Circuito Hamiltoniano (HCP)
- Problema de Localização de Facilidades (FLP)
- Problema de Roteamento de Veículos (Capacitados, Janela Tempo) (VRP, CVRP, VRPTW)
- Problema de Escalonamento de Projetos com Restrição de Recursos (RCPSP)
- Problema de Caminho Mínimo com Janelas de Tempo (SPTW)
- Problema de Isomorfismo de Grafos (GIP)
- Problema do Conjunto Dominante (DSP)
- Problema da k-Árvore Geradora Mínima (k-MST)
- Problema do Corte Máximo (MCP)
- Problema da Árvore Geradora Mínima Generalizada (GMST)
- Problema do Carteiro Chinês (CPP)
- Problema do Caminho Simples Mais Longo (LPP)
- Problema do k-Means e k-Clustering
- Problema do Bin-Packing (BPP)
- Problema de Escalonamento de Multiprocessadores (MSP)

[2] Crescenzi e Kann, *A compedium of NP Optimization Problems*

Referências Bibliográficas

Referências Bibliográficas

- CAMPELLO R. E, MACULAN N.
  Algoritmos e Heuristicas: Desenvolvimento, Avaliação e Performance.
  Editora da Universidade Federal Fluminense.
- NETTO P. O. B.
  Grafos: Teoria, Modelos e Algoritmos, 4a. edição.
  Editora Blucher. ISBN: 9788521203919
- GLOVER F., KOCHENBERG G. A.
  Handbook of Metaheuristics.
  Editora Springer. ISBN: 1-4020-7263-5
- GAREY M. R., JOHNSON D. S.
  Computers and Intractability - A Guide to the Theory of NP-Completeness.
  Editora Freeman and Company.
- GOLDBARG M. C., LUNA H. P.
  Otimização Combinatória e Programação Linear: Modelos e Algoritmos.
  Editora Campus.
- EIBEN A., SMITH J.
  Introduction to Evolutionary Computing.
  Editora Springer (Natural Computing Series). ISBN: 3540401849.
- PARDALOS P., RESENDE M. G.
  Handbook of Applied Optimization.
  Editora Oxford.

Referências Bibliográficas

RIVEST R. L., LEIRSON C. E., CORMEN, T. H., STEIN, C.
Algoritmos: Teoria e Prática, 3a. edição.
Editora Campus. ISBN: 9788535236996

LOPES, H. S., RODRIGUES L. C. A., STEINER M. T. A.
Meta-heurísticas em Pesquisa Operacional
Editora Ominipax. ISBN: 978-85-64619-10-4 [recurso eletrônico]
DOI: 10.7436/2013.mhpo.0