Metaheurísticas

Gustavo Peixoto Silva

Metaheurísticas

As **metaheurísticas** são procedimentos que guiam outras **heurísticas, ou seja, procedimentos computacionais**, usualmente de busca local, explorando o espaço de soluções além do **ótimo local**.

As metaheurísticas consideram boas características das soluções encontradas para explorar novas regiões promissoras.

Metaheurísticas

Os métodos exatos como o **Simplex** e os **Algoritmos de Fluxo em Redes** são capazes de encontrar ***a solução ótima*** de um determinado modelo.

As **metaheurísticas** podem encontrar a solução ótima de um determinado modelo, mas NÃO temos a garantia que a solução seja **A** ótima.

As metaheurísticas são usada somente quando o problema é **tão grande que o tempo de processamento do Simplex torna-se infinito** em relação à aplicação desejada. Problemas deste tipo são ditos *NP-hard* ou *NP-completo*.

Metaheurísticas

Partem de uma ou mais soluções iniciais e tentam melhorar a(s) solução(ões) por meio de modificações realizada(s) na(s) mesma(s).

São necessários os elementos:

1. Procedimento para gerar a solução inicial
2. Função objetivo ou função de avaliação
3. Método de modificação da(s) solução(ões) corrente(s) para encontrar uma solução que pode ser melhor do que a melhor solução conhecida.

Métodos heurísticos

- Construtivos

  Constroem uma solução passo a passo, introduzindo um elemento da solução a cada passo.

  Fornece uma Solução Inicial

- Refinamento ou Busca Local

  Tenta melhorar uma solução, através de modificações nos elementos da solução corrente.

Métodos Heurísticos Construtivos

Constroem uma solução passo a passo, introduzindo um elemento da solução a cada passo.

Fornece uma Solução Inicial para a Metaheurística

Heurística Construtiva Gulosa

- Constrói uma solução elemento a elemento
  - A cada passo é adicionado um único elemento
  - O elemento escolhido é o “melhor” segundo a função objetivo/função de avaliação
  - O método termina quando todos os elementos foram inseridos na solução

Problema de Designação (como o método guloso pode falhar!)

Três pessoas devem fazer três tarefas distintas, sendo que cada pessoa apresenta uma habilidade para realizar cada uma das tarefas. Cada tarefa é realizada por uma única pessoa e cada pessoa realiza uma única tarefa. Os dados são apresentados na tabela abaixo.

	T1	T2	T3
P1	13	10	9
P2	12	5	6
P3	8	6	10

	T1	T2	T3	Habilidade
P1	1	0	0	13
P2	0	0	1	6
P3	0	1	0	6 = 25

Problema de Designação

Três pessoas devem fazer três tarefas distintas, sendo que cada pessoa apresenta uma habilidade para realizar cada uma das tarefas. Cada tarefa é realizada por uma única pessoa e cada pessoa realiza uma única tarefa. Os dados são apresentados na tabela abaixo.

	T1	T2	T3
P1	13	10	9
P2	12	5	6
P3	8	6	10

	T1	T2	T3	Habilidade
P1	0	1	0	10
P2	1	0	0	12
P3	0	0	1	10 = 32

Exemplo da heurística construtiva gulosa

Seja, então, uma mochila de capacidade $b = 23$ e os 5 objetos da tabela abaixo, com os respectivos pesos e benefícios.

Objeto (j)	1	2	3	4	5
Peso (w_j)	4	5	7	9	6
Benefício (p_j)	2	2	3	4	4

Construamos uma solução para esse problema usando a seguinte idéia: adicionemos à mochila a cada passo, o objeto mais valioso por unidade de peso e que não ultrapasse a capacidade da mochila. Reordenando os objetos de acordo com a relação p_j/w_j, obtemos:

Objeto (j)	5	1	4	3	2
Peso (w_j)	6	4	9	7	5
Benefício (p_j)	4	2	4	3	2
(p_j/w_j)	0.67	0.50	0.44	0.43	0.40

Representemos uma solução s por um vetor binário de n posições.

Exemplificando a aplicação da heurística construtiva gulosa

Passo 1 : Adicionemos, primeiramente, o objeto 5, que tem a maior relação p_j/w_j

$s = (00001)^t$

$f(s) = 4$

Peso corrente da mochila $= 6 <$ b $= 23$

Passo 2 : Adicionemos, agora, o objeto 1, que tem a segunda maior relação p_j/w_j

$s = (10001)^t$

$f(s) = 6$

Peso corrente da mochila $= 10 <$ b $= 23$

Passo 3 : Adicionemos, agora, o objeto 4, que tem a terceira maior relação p_j/w_j

$s = (10011)^t$

$f(s) = 10$

Peso corrente da mochila $= 19 <$ b $= 23$

Passo 4 : O objeto a ser alocado agora seria o terceiro. No entanto, esta alocação faria superar a capacidade da mochila. Neste caso, devemos tentar alocar o próximo objeto com a maior relação p_j/w_j, que é o objeto 1. Como também a alocação desse objeto faria superar a capacidade da mochila e não há mais objetos candidatos, concluímos que a solução anterior é a solução final, isto é: $s^* = (10011)^t$ com $f(s^*) = 10$.

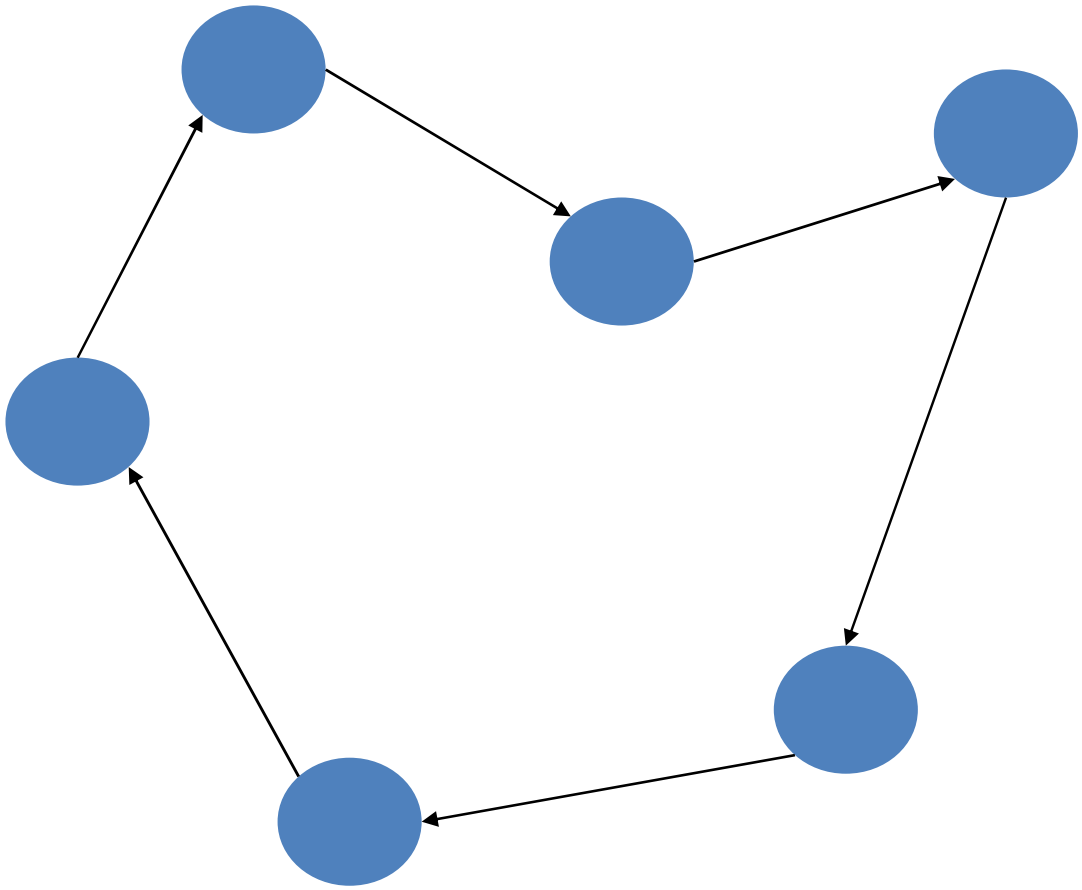
Problema do Caixeiro Viajante
Travelling sales problem

- Dado de entrada: conjunto de cidades e uma matriz de distâncias entre elas

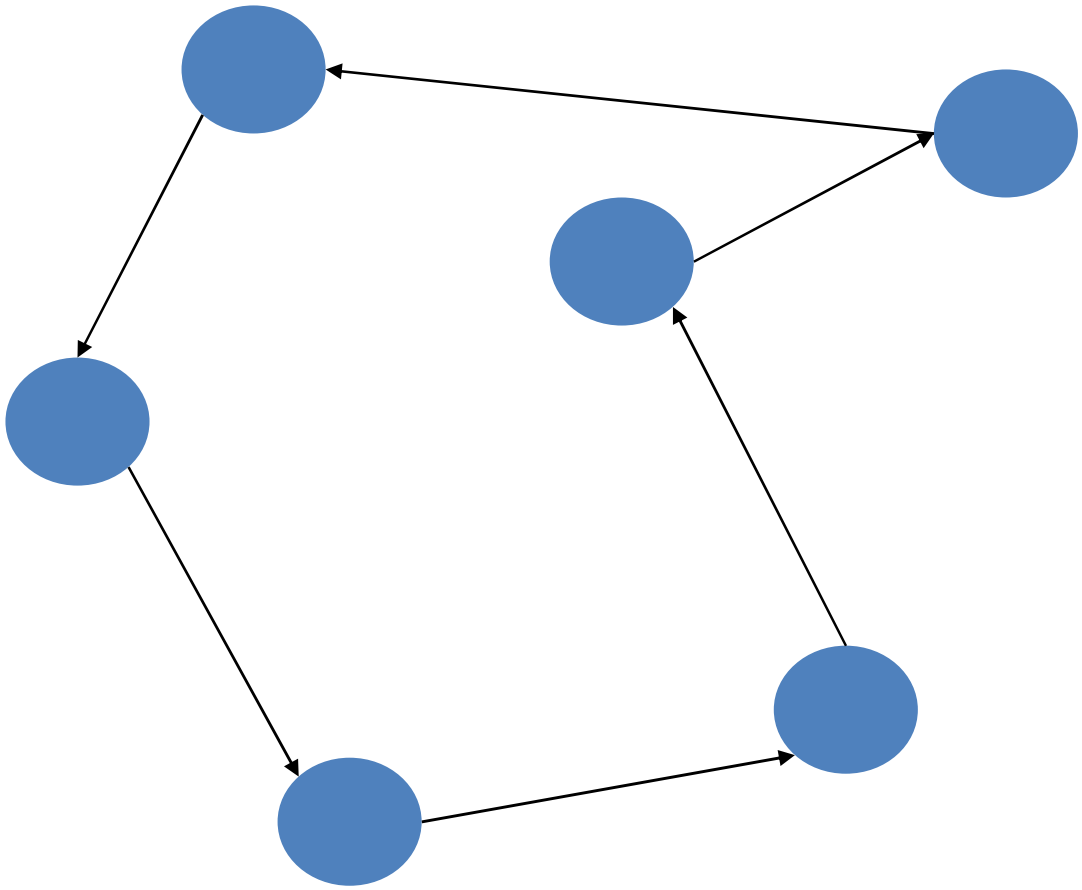
- PCV (TSP) consiste em encontrar uma rota que:
  - parte de uma cidade origem
  - passe por todas as demais cidades uma única vez
  - retorne à cidade origem ao final do percurso
  - percorra a menor distância possível

- Esta rota é conhecida como Ciclo Hamiltoniano

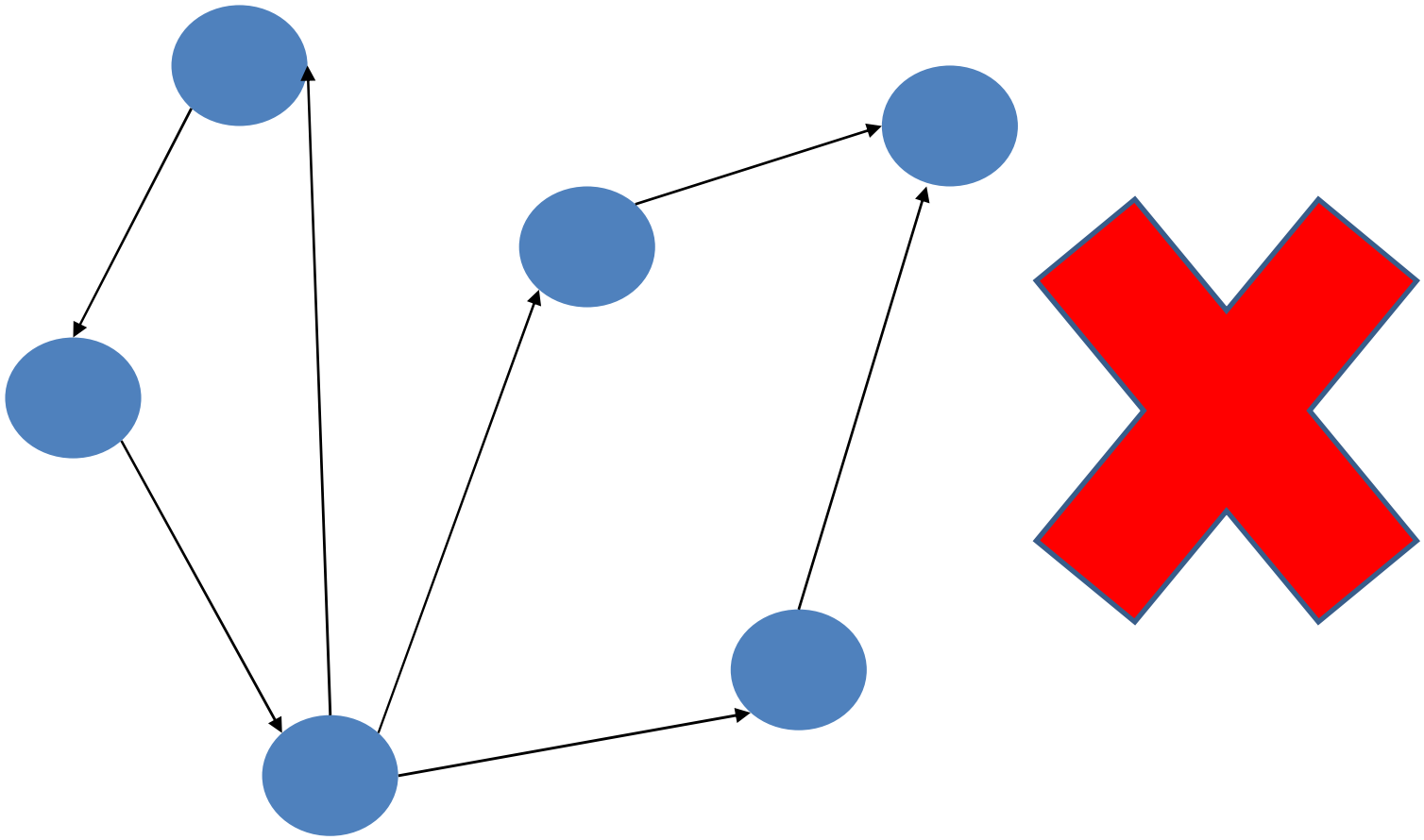
Problema do Caixeiro Viajante



Problema do Caixeiro Viajante



Problema do Caixeiro Viajante



Estas duas rotas não são uma solução para o PCV

Heurísticas construtivas para o PCV

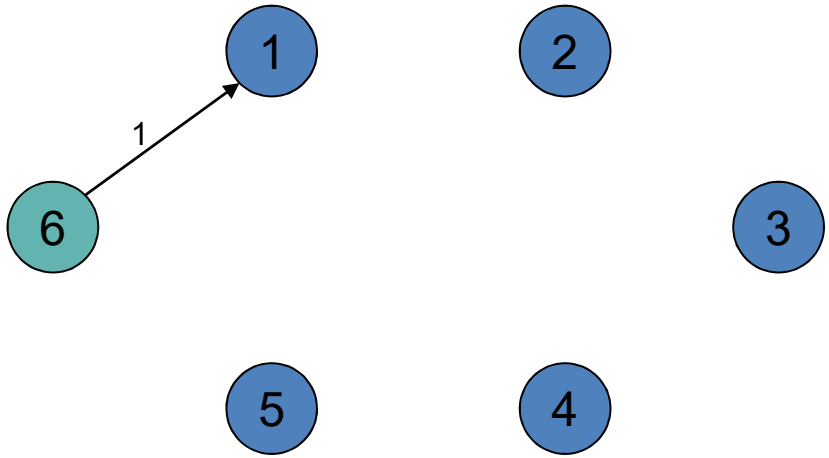
- Vizinho mais próximo
  - Construir uma rota adicionando, a cada passo, a cidade mais próxima da última cidade inserida e que ainda não foi visitada.

- Inserção mais barata
  - Construir uma rota, partindo de rota uma inicial envolvendo 3 cidades e adicionar a cada passo, a cidade *k* (não visitada) entre a ligação (*i, j*) de cidades já visitadas, cujo custo de inserção seja o mais barato

PCV – Vizinho mais Próximo partindo do nó 6
Exemplo - Passo 1

Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

i	j	d_{ij}
6	**1**	1
6	**2**	2
6	**3**	6
6	**4**	6
6	**5**	2

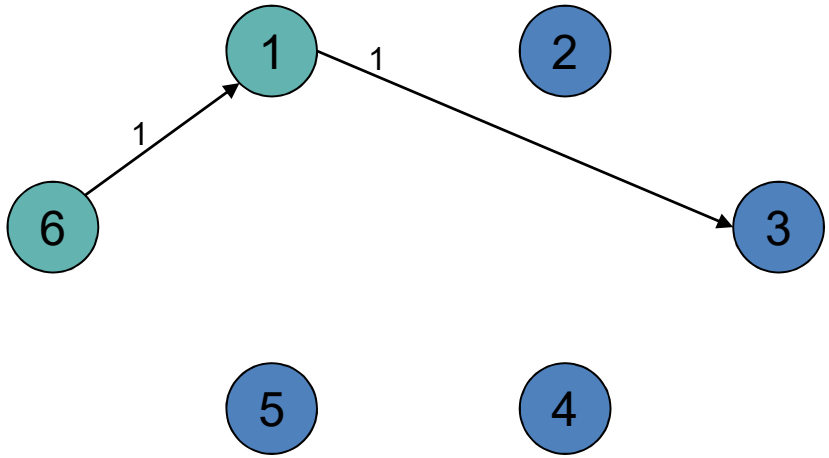


- **Distância Total =** 1

PCV – Vizinho mais Próximo
Exemplo - Passo 2

Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

i	j	d_{ij}
1	**2**	2
1	**3**	1
1	**4**	4
1	**5**	9

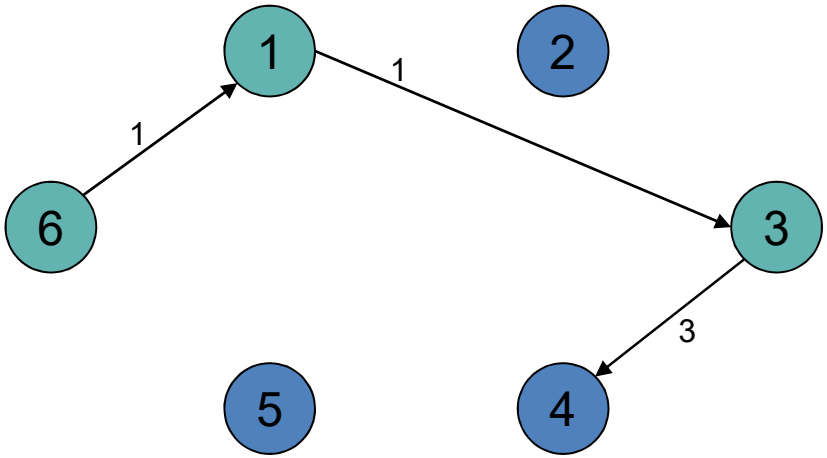


- **Distância Total =** 1 + 1 = 2

PCV – Vizinho mais Próximo
Exemplo - Passo 3

Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

i	j	d_{ij}
3	**2**	5
3	**4**	3
3	**5**	8

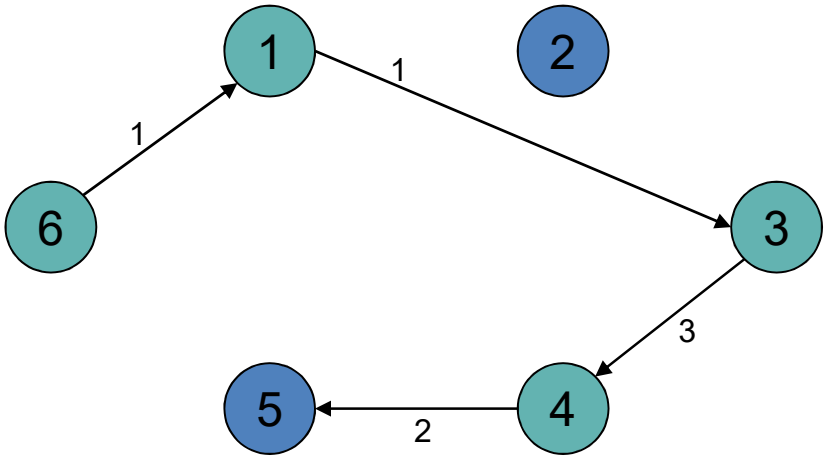


- **Distância Total =** 2 + 3 = 5

PCV – Vizinho mais Próximo
Exemplo - Passo 4

Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

i	j	d_{ij}
4	**2**	9
4	**5**	2

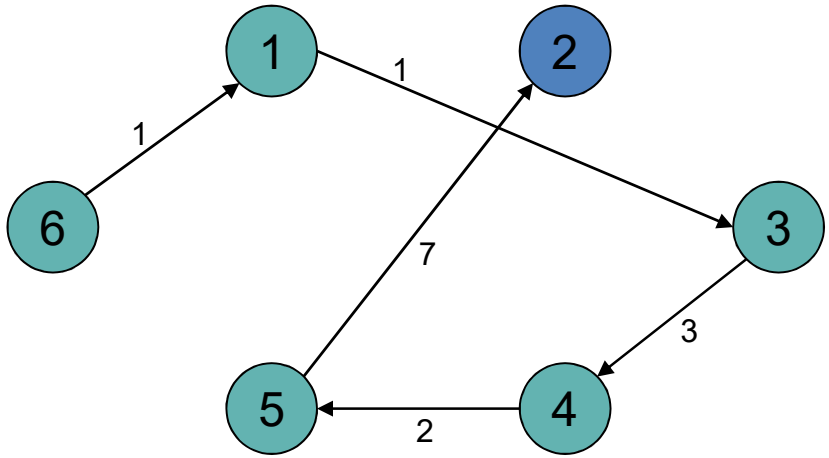


- **Distância Total =** 5 + 2 = 7

PCV – Vizinho mais Próximo
Exemplo - Passo 5

i	j	d_{ij}
5	2	7

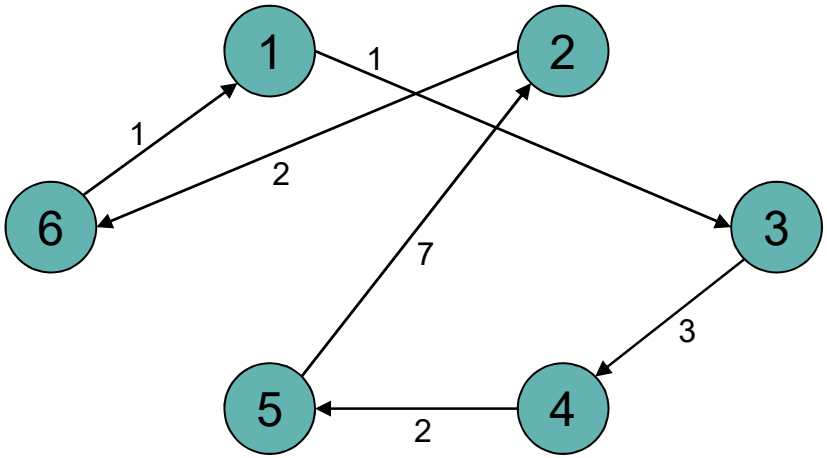
Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0



- **Distância Total =** 7 + 7 = 14

PCV – Vizinho mais Próximo
Exemplo – Passo final: "Inserção forçada"

Cid.	1	2	3	4	5	6
1	0	2	1	4	9	1
2	2	0	5	9	7	2
3	1	5	0	3	8	6
4	4	9	3	0	2	6
5	9	7	8	2	0	2
6	1	2	6	6	2	0

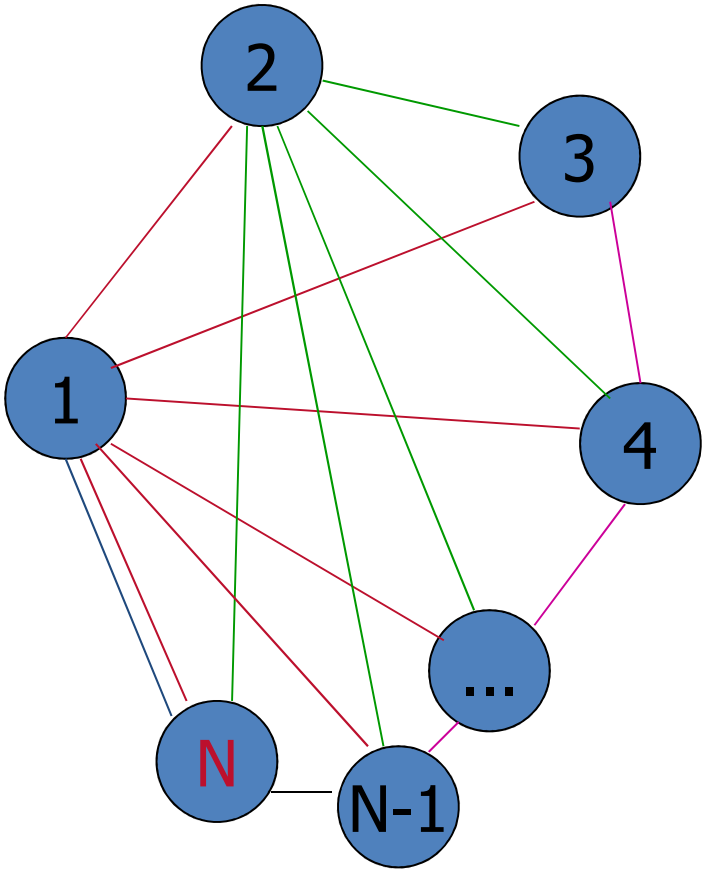


- **Distância Total =** 14 + 2 = 16

Representação da solução

s = (6 1 3 4 5 2)

Complexidade da heurística construtiva do vizinho mais próximo aplicada ao PCV



Iteração	Número de avaliações
1	N-1
2	N-2
...	...
N-1	1
N	1
Total	1 + N(N-1)/2

Problemas Combinatoriais

São problemas cujas variáveis de decisão devem ser inteiras positivas ou binárias (0/1).

As soluções são combinações factíveis dos dados de entrada do problema

O Problema de Sequenciamento de Tarefas, Programação da mão de obra, Designação, da Mochila e o PCV são tipicamente combinatoriais, assim como muito outros que já foram estudados.

Problema do Caixeiro Viajante: Complexidade

- Considerando PCV simétrico ($d_{ij} = d_{ji}$), para *n* cidades há (*n*-1)!/2 rotas possíveis
- Para *n* = 20 cidades, são 10^{16} rotas possíveis. Um computador que avalia uma rota em 10^{-8} segundos gastaria cerca de 19 anos para encontrar a melhor solução por enumeração completa!
- Métodos de enumeração implícita, tais como *branch-and-bound*, podem reduzir significativamente este tempo
- Mas não há garantia de que isso sempre ocorra

Complexidade do Problema do Caixeiro Viajante

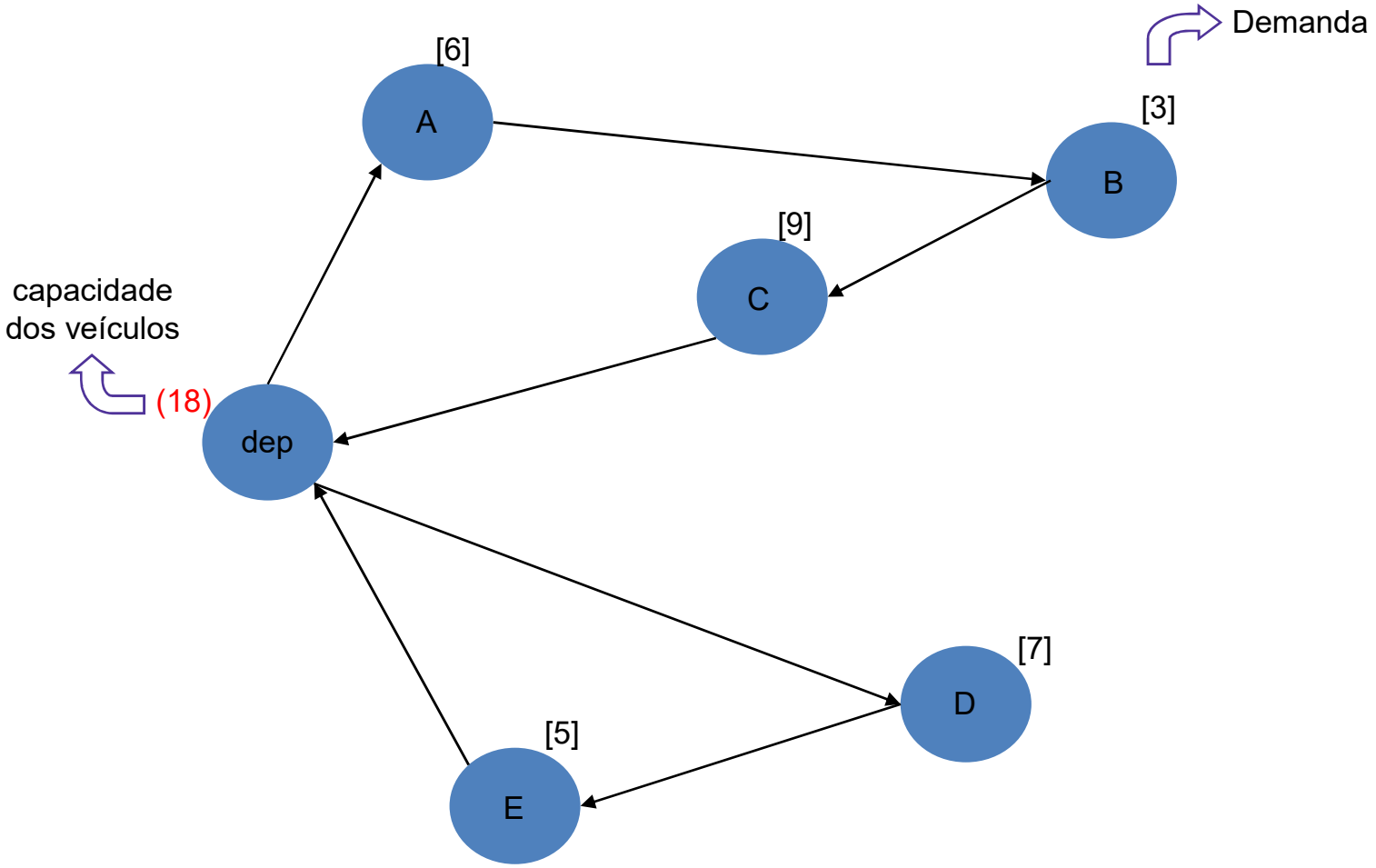
- Para dimensões mais elevadas, a resolução por métodos de programação matemática é proibitiva em termos de tempos computacionais
- PCV é da classe NP-difícil: não existem algoritmos exatos que o resolvam em tempo polinomial
- À medida que *n* cresce, o tempo cresce exponencialmente
- PCV é resolvido por meio de heurísticas:
  - Procedimentos que seguem uma intuição para resolver o problema
  - Não garantem a otimalidade da solução final
  - Em geral, produzem soluções de boa qualidade rapidamente

Problema de Roteamento de Veículos

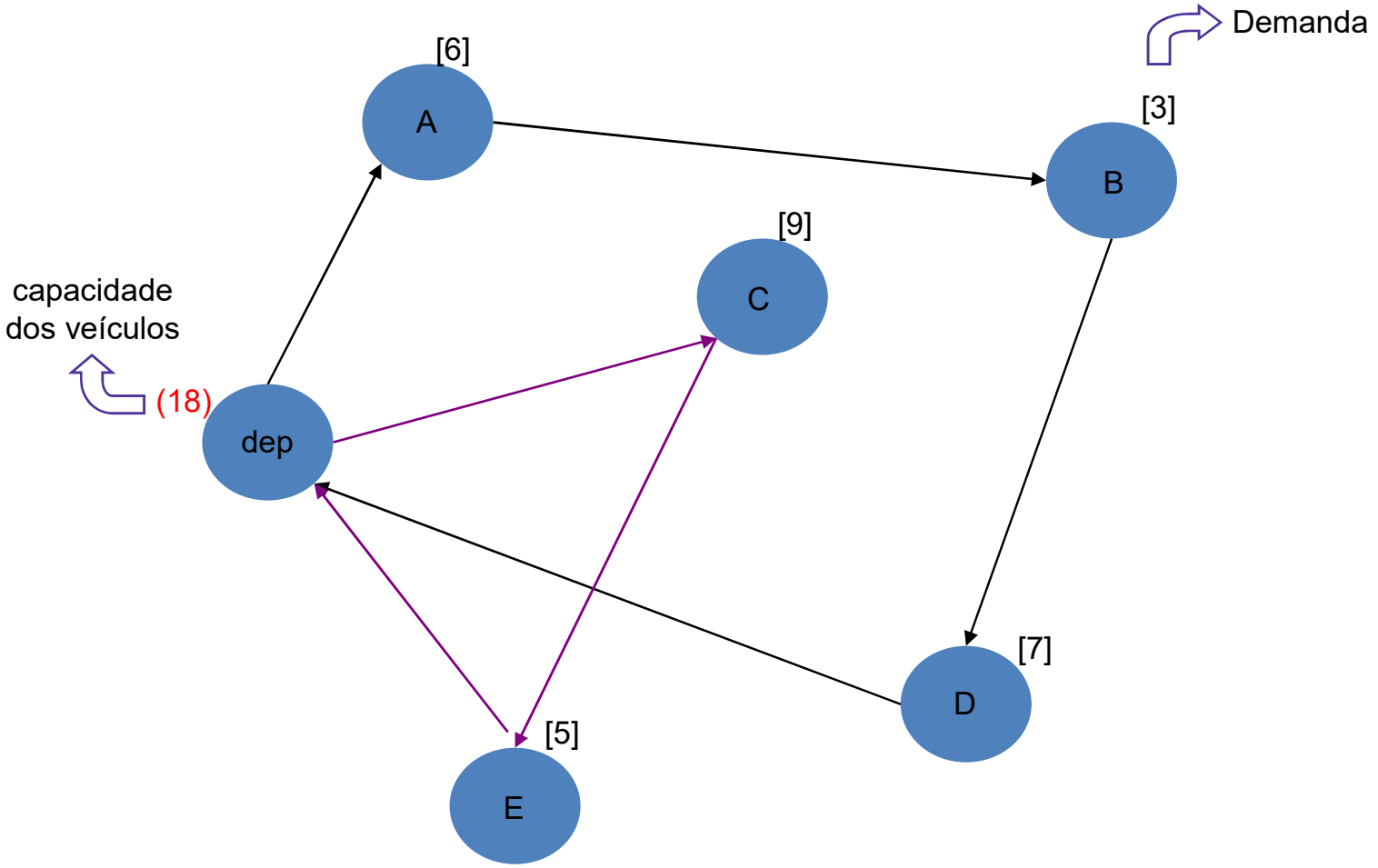
- Dados de entrada:
  - Um depósito
  - Uma frota de veículos, com base no depósito
  - Um conjunto de clientes
  - A demanda dos clientes
  - Uma matriz de distâncias D = (d_{ij}) entre depósito e clientes e entre pares de clientes
  - Cidades = 1 depósito + n clientes

- PRV consiste em encontrar um conjunto de rotas para os veículos tal que:
  - Cada rota comece e termine no depósito
  - Cada cliente seja atendido por um único veículo
  - A capacidade dos veículos seja respeitada
  - A distância total percorrida sejam a menor possível e/ou
  - O número de veículos seja minimizado

Problema de Roteamento de Veículos



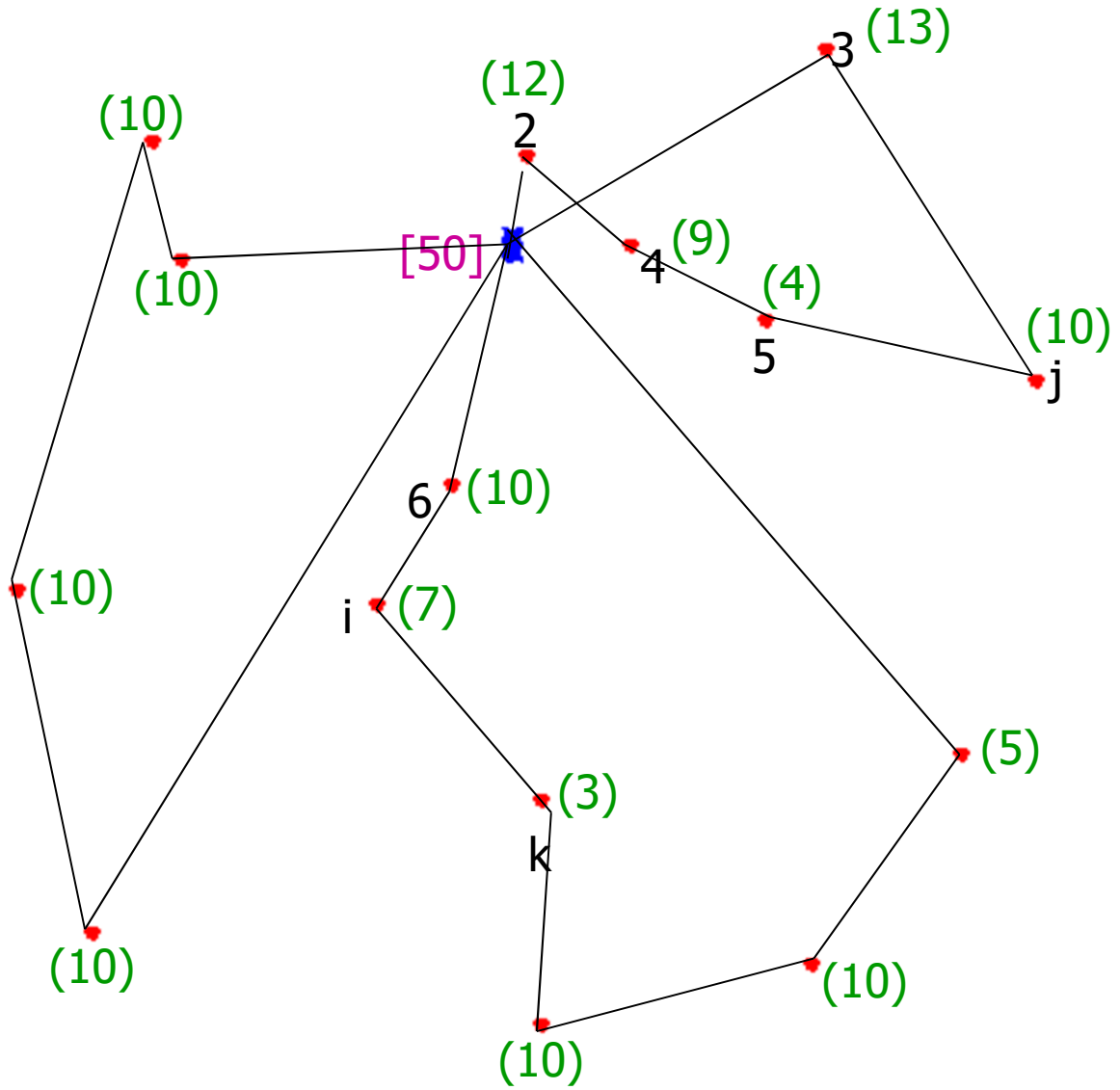
Problema de Roteamento de Veículos



Adaptação da Heurística do Vizinho mais próximo para o Problema de Roteamento de Veículos

- Passo 1: Partir do depósito com um novo veículo e ir até a cidade mais próxima ainda não visitada;
- Passo 2: Determinar a cidade mais próxima da última cidade inserida na rota e verificar se é possível atender sua demanda;
- Passo 3: Se for possível atender a demanda dessa cidade, adicioná-la à rota. Caso contrário, retornar ao depósito e voltar ao Passo 1.
- Passo 3': Se for possível atender a demanda dessa cidade, adicioná-la à rota. Caso contrário, tentar a próxima cidade mais próxima. Se não houver mais cidade, retornar ao depósito e voltar ao Passo 1.

Heurística Construtiva do Vizinho mais Próximo Aplicada ao PRV



Problema de Programação de uma Máquina

Dado um conjunto de tarefas $N = \{1, 2, ..., n\}$, para cada $j \in N$ temos:

1. p_j - *tempo de processamento*,
2. d_j - *data de entrega*,
3. w_j - *peso por atraso na entrega*
4. C_j - *tempo de completude* ~ *variável de decisão*

As tarefas devem ser processadas em uma sequencia ***S*** em uma máquina minimizando T(S) dado por:

$$T(S) = \sum_{j=1}^{n} w_j \times \max\{C_j - d_j, 0\}$$

Problema de Programação de Máquinas

Job j	1	2	3	4	5	6
Processing time p_j	3	1	1	5	1	5
Weight w_j	3	5	1	1	4	4
Due date d_j	1	5	3	1	3	1

Inserir na máquina a tarefa com menor data de entrega (d_j).
Em caso de empate, inserir a tarefa com menor tempo de processamento (p_j)
Em caso de empate novamente, inserir a tarefa com menor peso por atraso (w_j)

Job j	1	4	6	3	5	2
Pj	3	5	5	1	1	1
Wj	3	1	4	1	4	5
Cj	3	8	13	14	15	16
Dj	1	1	1	3	3	5
Atraso	3 - 1 = 2	8 - 1 = 7	13 – 1 = 12	14 – 3 = 11	15 – 3 = 12	16 – 5 = 11
Custo	2*3 = 6	7*1 = 7	12*4 = 48	11*1 = 11	12*4 = 48	11*5 = 55

Custo total = 175