Heurísticas de refinamento: Fundamentação

Marcone Jamilson Freitas Souza[1,2,3]

Puca Huachi Vaz Penna[1]

[1] Departamento de Computação

[1] Programa de Pós-Graduação em Ciência da Computação

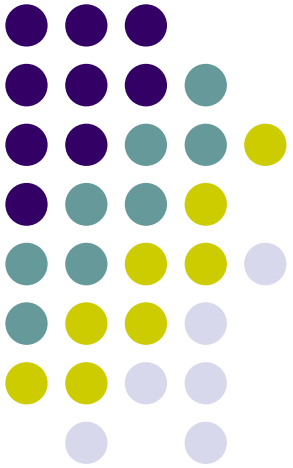
Universidade Federal de Ouro Preto

[2] Programa de Pós-graduação em Modelagem Matemática e Computacional / CEFET-MG

[3] Programa de Pós-graduação em Instrumentação, Controle e Automação de Processos de Mineração / ITV/UFOP

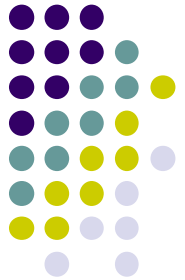
www.decom.ufop.br/prof/marcone, www.decom.ufop.br/puca

E-mail: {marcone,puca}@ufop.edu.br



Souza, Marcone J. F. e Penna, Puca H. V. Heurísticas de refinamento: fundamentação. Notas de aula de Técnicas Metaheurísticas para Otimização Combinatória. Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, 2021. Disponível em www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/HeuristicasRefinamento.pptx

Sumário



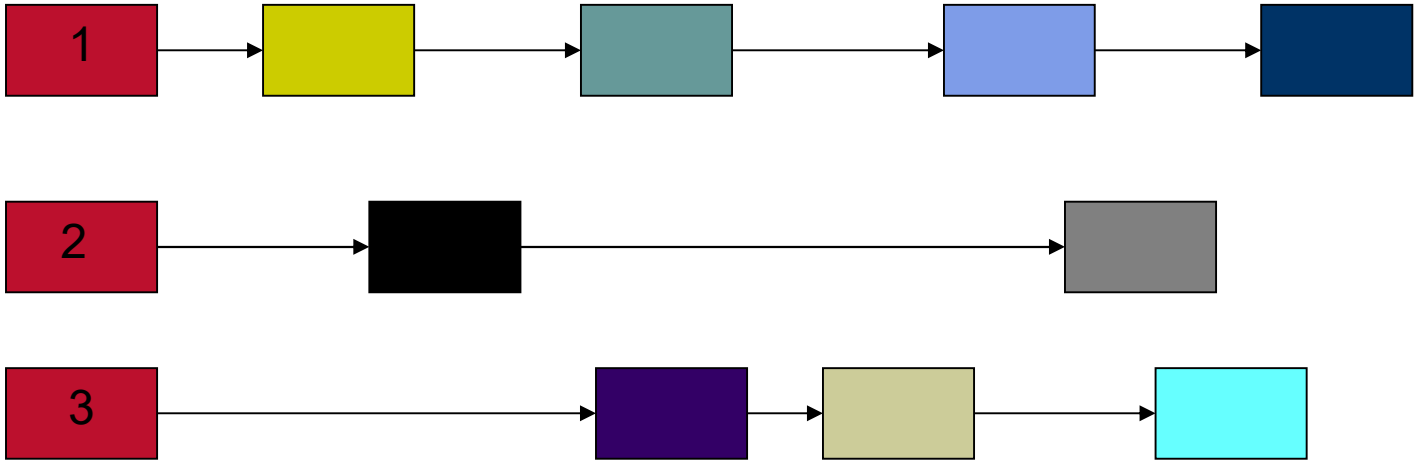
- Estruturas de vizinhança:
  - Definição de movimento
  - Exemplos:
    - Programação de tripulações (*Crew Scheduling*)
    - Problema da Mochila (*Knapsack problem*)
    - Problema do Caixeiro Viajante (*Traveling Salesman Problem*)
- Métodos de busca local:
  - Método da Descida (*Descent Method*)
  - Método da Subida (*Uphill method*)

Heurísticas de refinamento

- Também chamadas de heurísticas de busca local
- Baseadas na noção de vizinhança
- A cada solução *s* está associado um conjunto de vizinhos *s'* definidos por um determinado tipo de movimento *m*
- $s' = s \oplus m$
- O tipo de movimento é dependente do problema

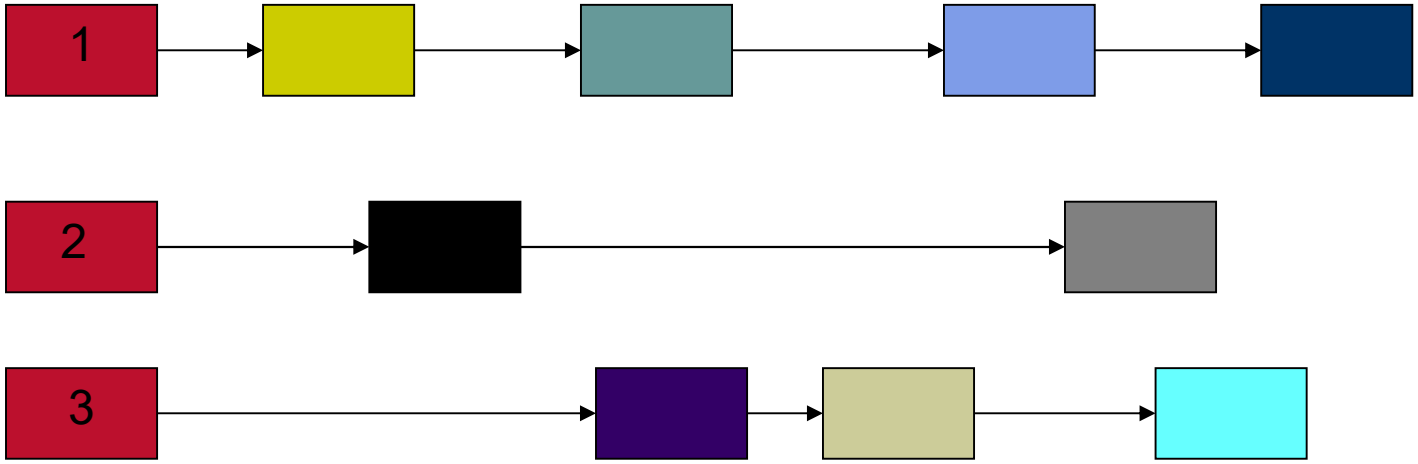
Programação de Tripulações (*Crew Scheduling*)

- Dado um conjunto de **n** tarefas (viagens), alocá-las a um conjunto de **m** tripulações, de forma a atender restrições operacionais e legais e minimizar o custo com o uso das tripulações.



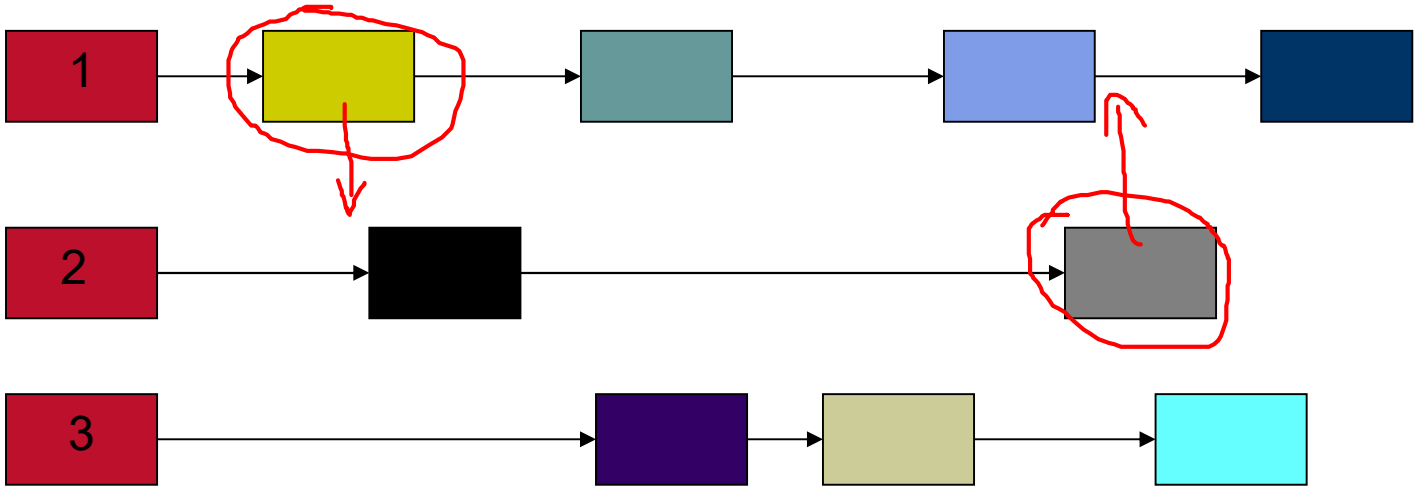
Programação de Tripulações (*Crew Scheduling*)

- Representação de uma solução:
  - Vetor s de **m** posições, em que cada posição indica uma tripulação e a ela está associada uma lista de tarefas a serem executadas por essa tripulação



Programação de Tripulações (*Crew Scheduling*)

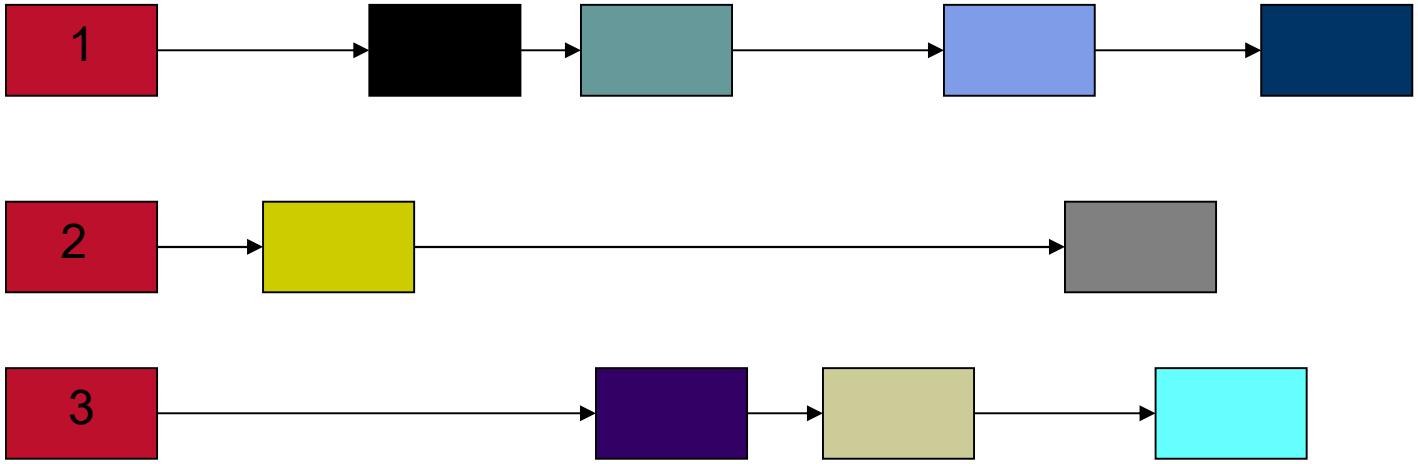
- Estruturas de vizinhança:
  - Movimento 1 (troca com disponibilidade): Trocar uma tarefa de uma tripulação com outra tarefa de outra tripulação que esteja disponível



Solução s

Programação de Tripulações (*Crew Scheduling*)

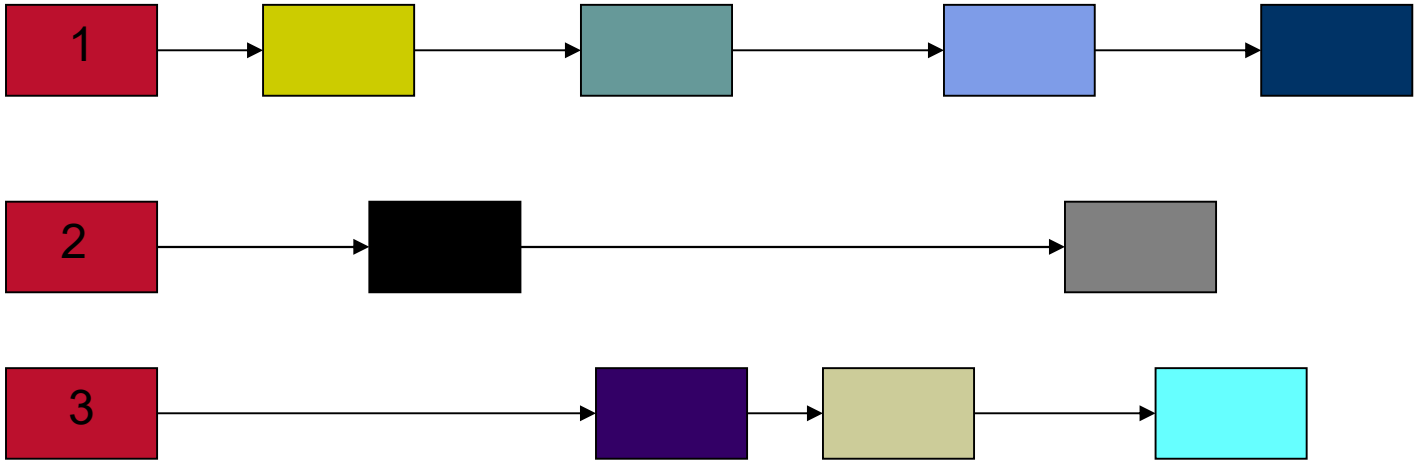
- Estruturas de vizinhança:
  - Movimento 1 (troca com disponibilidade): Trocar uma tarefa de uma tripulação com outra tarefa de outra tripulação que esteja disponível



Solução s’

Programação de Tripulações (*Crew Scheduling*)

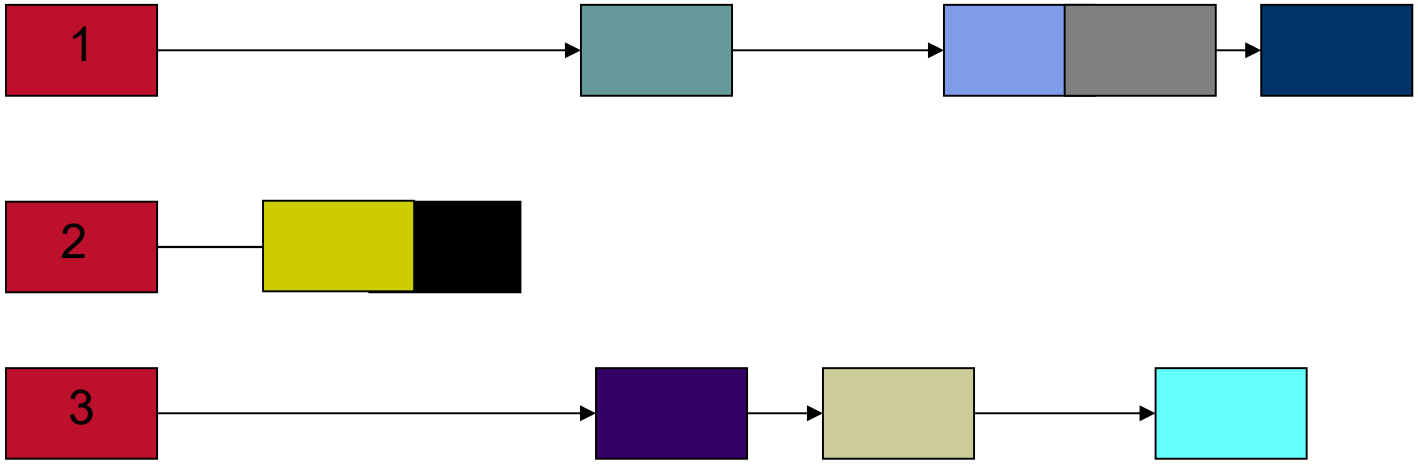
- Estruturas de vizinhança:
  - Movimento 2 (troca): Trocar uma tarefa de uma tripulação com outra tarefa de outra tripulação



Solução s

Programação de Tripulações (*Crew Scheduling*)

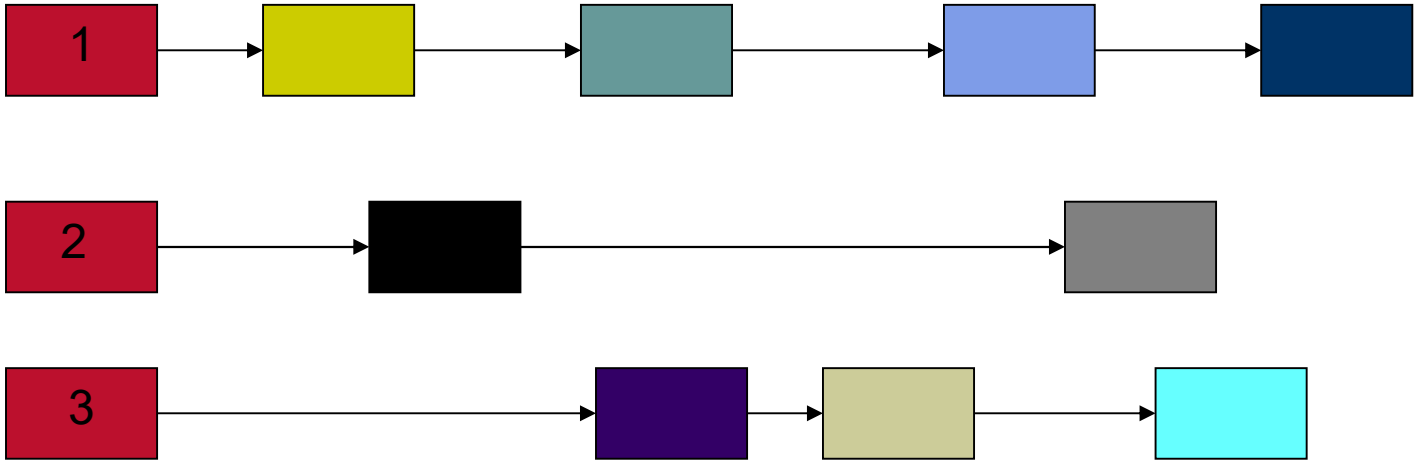
- Estruturas de vizinhança:
  - Movimento 2 (troca): Trocar uma tarefa de uma tripulação com outra tarefa de outra tripulação



Solução s’

Programação de Tripulações (*Crew Scheduling*)

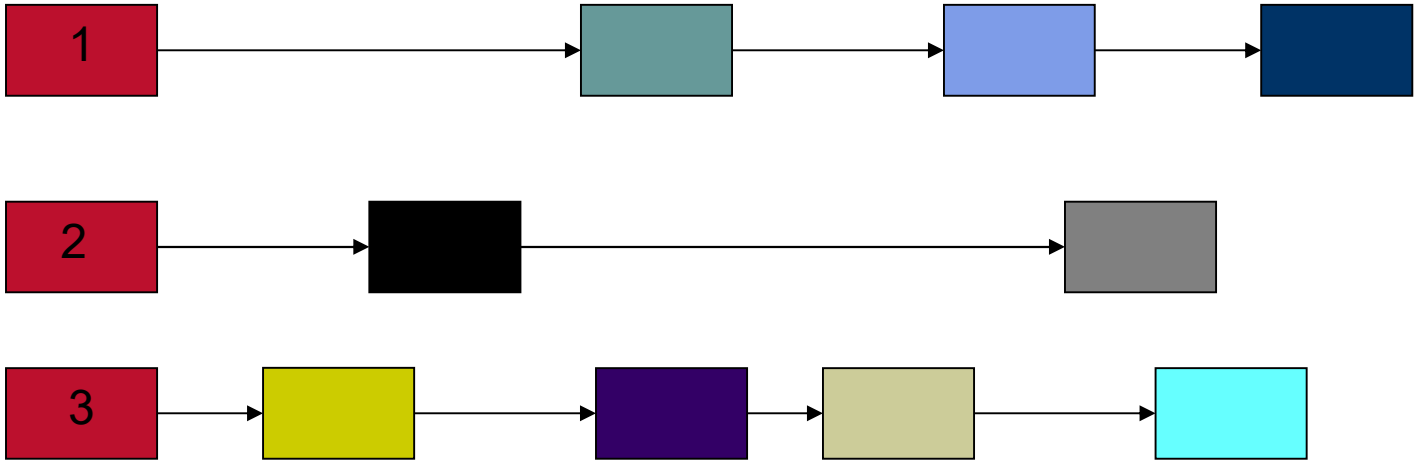
- Estruturas de vizinhança:
  - Movimento 3 (realocação com disponibilidade): Realocar uma tarefa de uma tripulação para outra tripulação que esteja disponível



Solução s

Programação de Tripulações (*Crew Scheduling*)

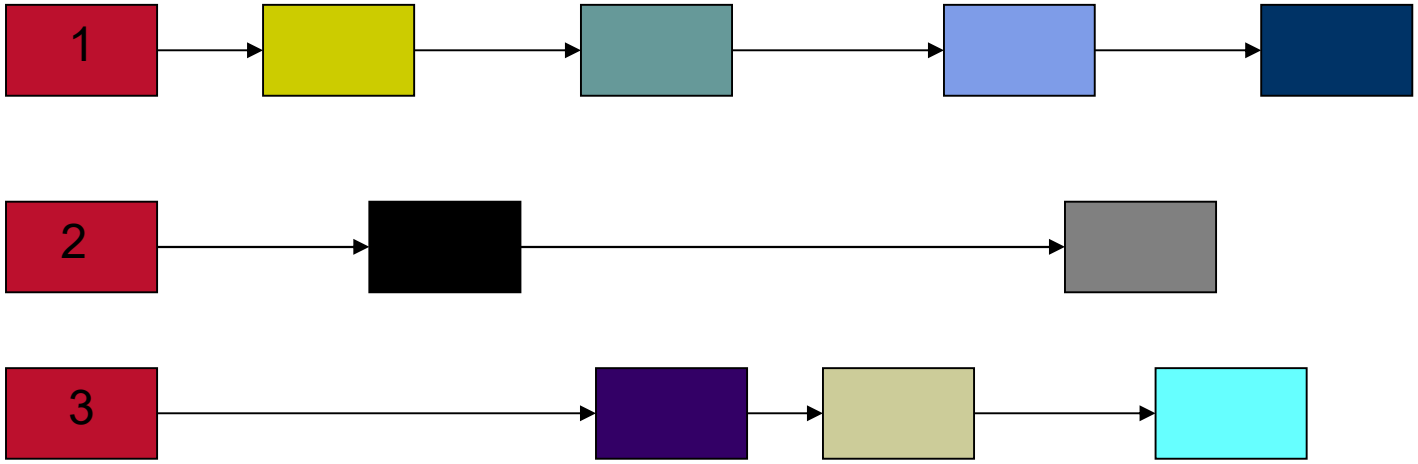
- Estruturas de vizinhança:
  - Movimento 3 (realocação com disponibilidade): Realocar uma tarefa de uma tripulação para outra tripulação que esteja disponível



Solução s’

Programação de Tripulações (*Crew Scheduling*)

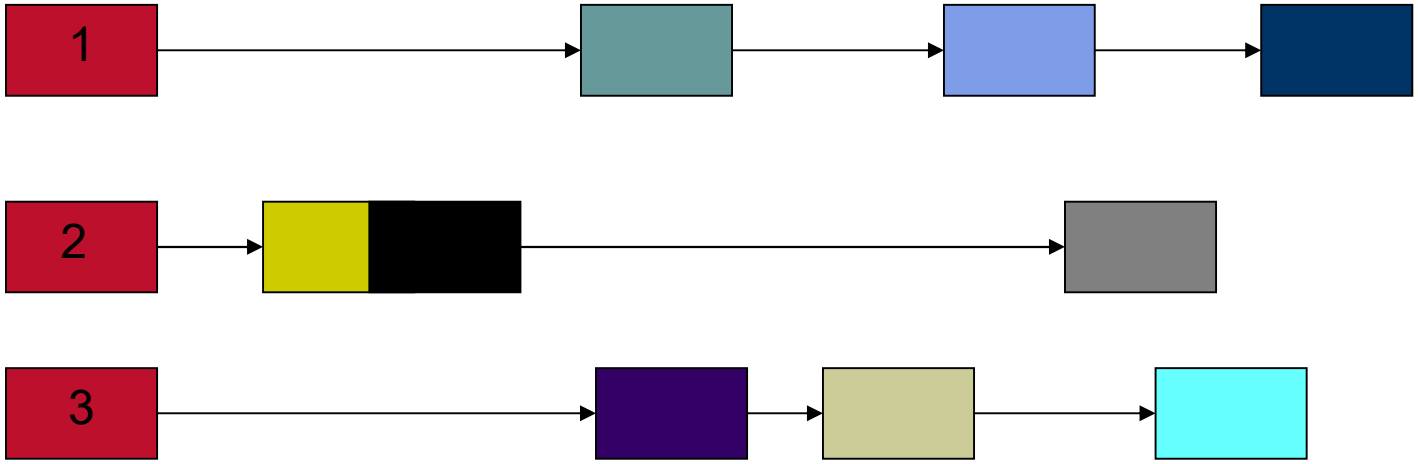
- Estruturas de vizinhança:
  - Movimento 4 (realocação): Realocar uma tarefa de uma tripulação para outra tripulação



Solução s

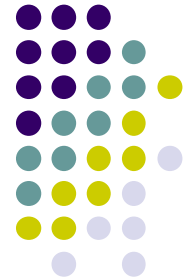
Programação de Tripulações (*Crew Scheduling*)

- Estruturas de vizinhança:
  - Movimento 4 (realocação): Realocar uma tarefa de uma tripulação para outra tripulação



Solução s’

Programação de Tripulações (*Crew Scheduling*)



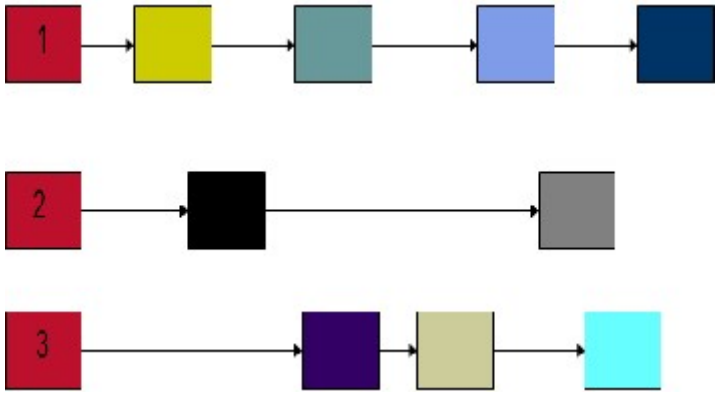
- Nos movimentos 2 (troca) e 4 (realocação), permite-se gerar soluções **inviáveis**
- Neste caso, a função de avaliação deve penalizar o tempo em que uma tripulação está executando mais de uma tarefa ao mesmo tempo
- Questionamento:
  - Com qual movimento (ou quais movimentos) é possível explorar todo o espaço de busca?
  - Isto é, dada uma solução, com qual ou quais movimentos podemos alcançar qualquer outra solução do espaço de busca?

Programação de Tripulações (*Crew Scheduling*)

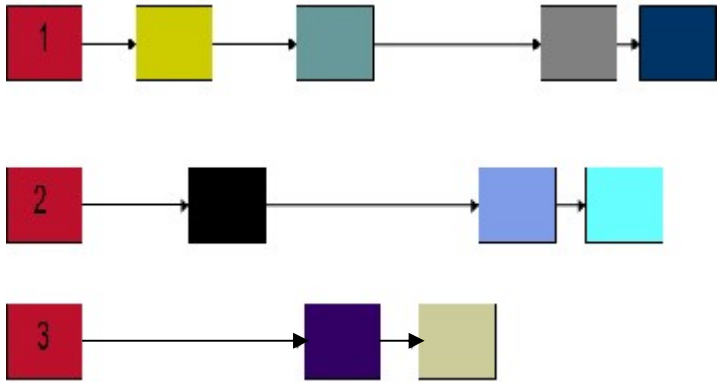
- Nos movimentos de troca de tarefas (com ou sem disponibilidade), o número de tarefas por tripulação permanece o mesmo após o movimento
  - Então não é possível explorar todo o espaço de busca com eles
- No movimento de realocação de uma tarefa (com disponibilidade) também não há garantia de se explorar todo o espaço de busca

Programação de Tripulações (*Crew Scheduling*)

- De fato, considerando-se as figuras abaixo, partindo-se da solução *s* **não é possível** alcançar a solução *s**, usando-se APENAS de movimentos de realocação (com disponibilidade):



Solução *s*



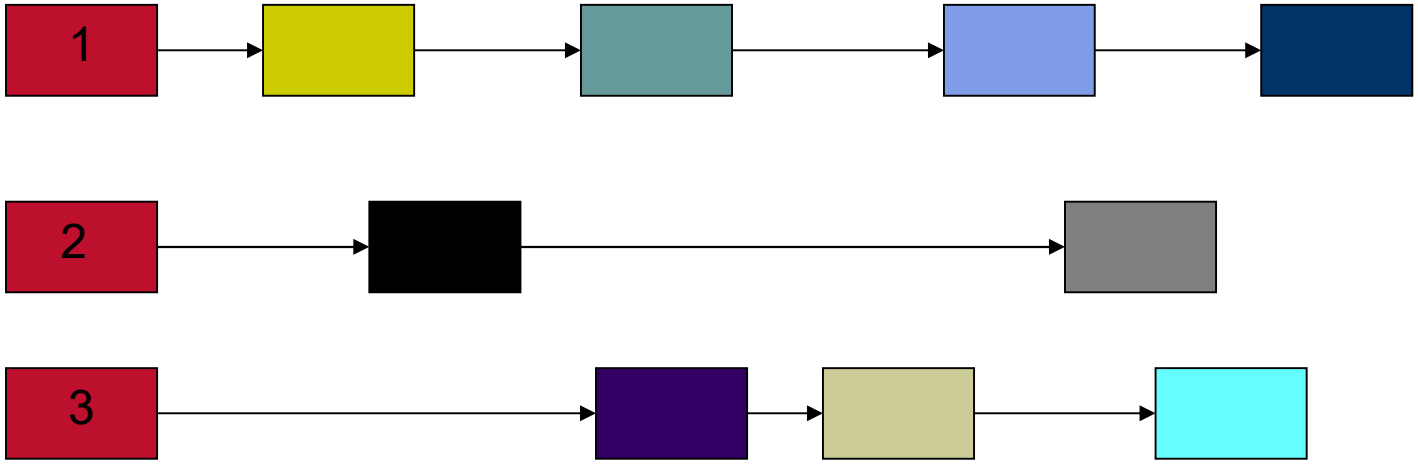
Solução *s**

Programação de Tripulações (*Crew Scheduling*)

- No movimento de realocação de uma tarefa (admitindo-se inviabilidade) é possível partir de uma dada solução e gerar **qualquer** outra após um número finito de operações com este movimento
  - Então este movimento é ESSENCIAL para explorar o espaço de soluções do problema

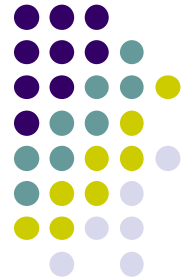
Programação de Tripulações (*Crew Scheduling*)

- De fato, partindo-se da solução s (abaixo) é possível se chegar à solução s*, após 3 aplicações do movimento de realocação (com inviabilidade):

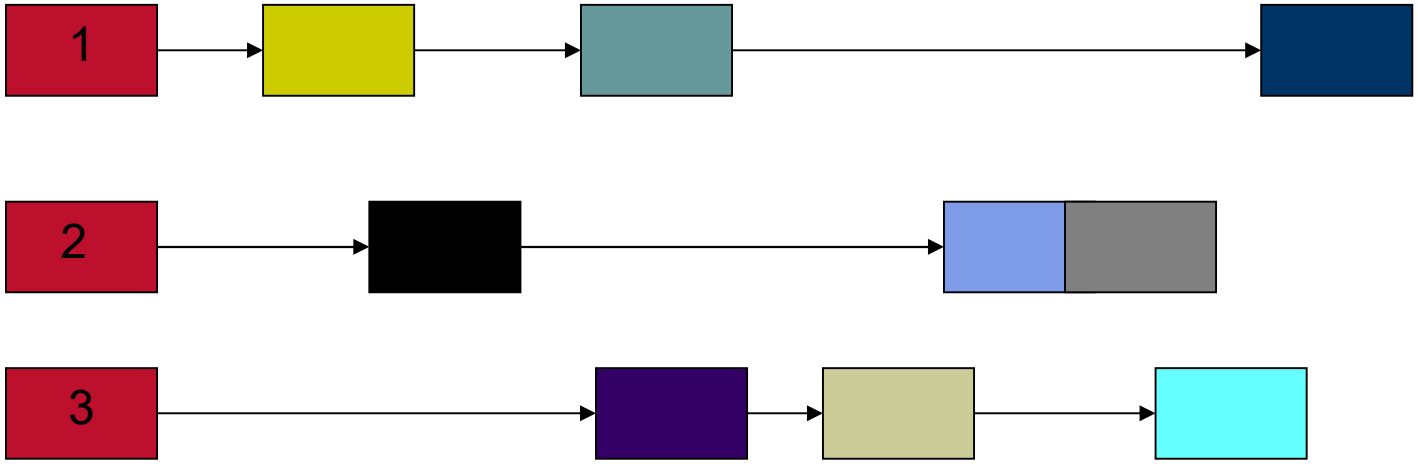


Solução s

Programação de Tripulações (*Crew Scheduling*)



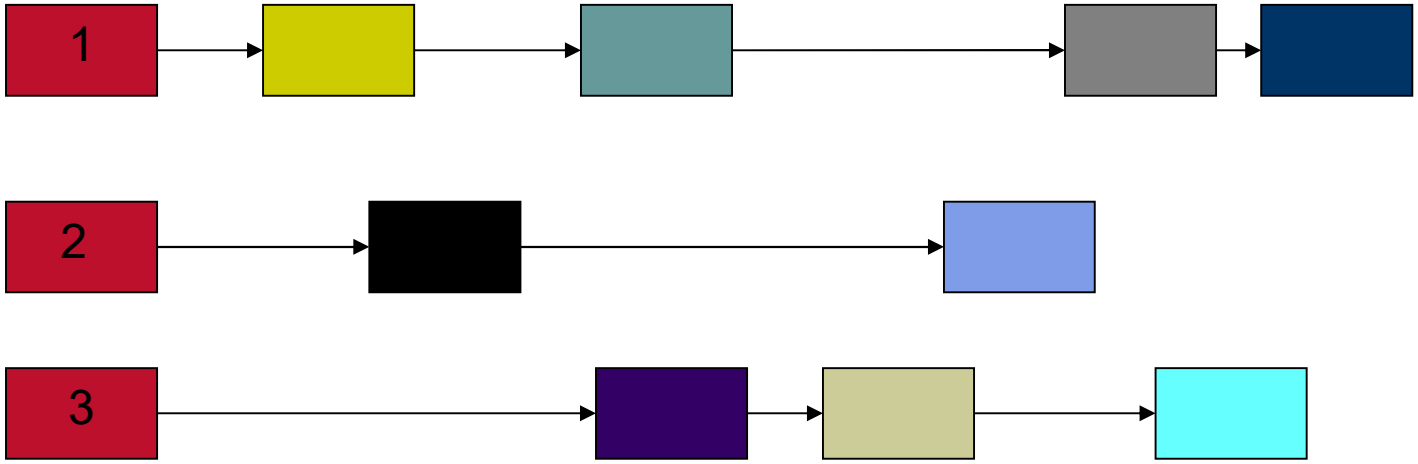
- 1ª Iteração:



Solução s’

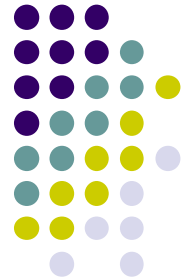
Programação de Tripulações (*Crew Scheduling*)

- 2ª Iteração:

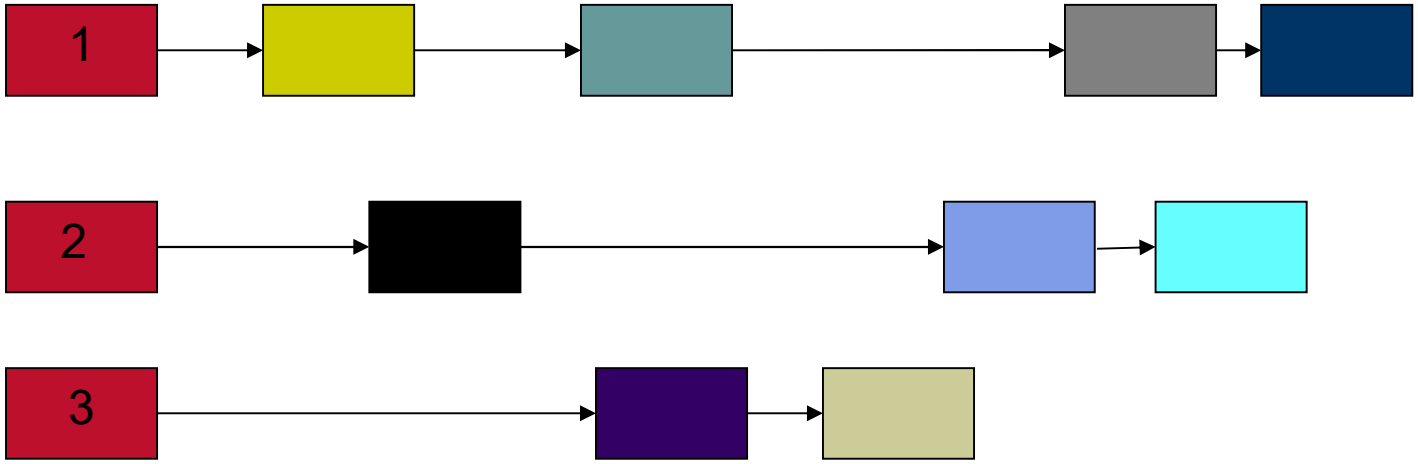


Solução s''

Programação de Tripulações (*Crew Scheduling*)



- 3ª Iteração:



Solução s*

Problema da Mochila

- Dado um conjunto de **n** itens, cada qual com seu peso $\mathbf{w_j}$ e retorno $\mathbf{p_j}$, escolher os itens a serem colocados em uma mochila de capacidade **b** tal que o retorno (benefício) dos itens carregados seja máximo
- Representação de uma solução:
  - Vetor s de **n** posições, em que cada posição **j** armazena o valor 1 se o item **j** for alocado à mochila e 0, caso contrário
  - $s = (s_1\ s_2\ s_3\ ...\ s_n)$, em que $s_j \in \{0,1\}\ \forall j = 1, ..., n$

Problema da Mochila

- Avaliação de uma solução:
  - Se for trabalhar no espaço de soluções **viáveis**:
    - A função de avaliação deve ser igual à função objetivo:
      - $f(s) = \sum_{j=1}^{n} p_j s_j$
  - Se for trabalhar no espaço de soluções **inviáveis**:
    - A função de avaliação deve penalizar o não atendimento à restrição de peso:
      - $f(s) = \sum_{j=1}^{n} p_j s_j - \alpha \times \underbrace{max\{0, \sum_{j=1}^{n} w_j s_j - b\}}_{\text{Excesso de peso}}$
      - α é a penalização por violar a capacidade da mochila
      - $\alpha = \sum_{j=1}^{n} p_j$

Problema da Mochila

- Supor uma mochila com capacidade b = 23 e os itens conforme tabela abaixo:

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

  - $f(s) = \sum_{j=1}^{n} p_j s_j - \alpha \times max\{0, \sum_{j=1}^{n} w_j s_j - b\}$
  - Seja a solução *s* = (1 0 1 0 1):
    - Soma dos benefícios: 2×1 + 2×0 + 3×1 + 4×0 + 4×1 = 9
    - Soma das capacidades: 4×1 + 5×0 + 7×1 + 9×0 + 6×1 = 17
    - $\alpha = \sum_{j=1}^{n} p_j$ = 2 + 2 + 3 + 4 + 4 = 15
    - A capacidade da mochila é respeitada, então:
    - $f(s) = \sum_{j=1}^{n} p_j s_j - \alpha \times max\{0, 17 - 23\}$
    - $f(s) = 9 - 15 \times max\{0, -6\}$ = 9

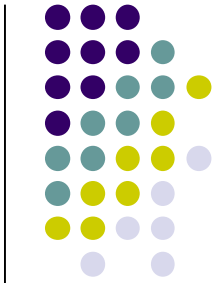
Problema da Mochila

- Supor uma mochila com capacidade b = 23 e os itens conforme tabela abaixo:

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

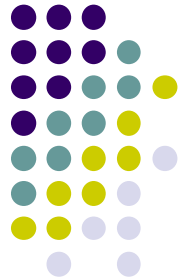
  - $f(s) = \sum_{j=1}^{n} p_j s_j - \alpha \times max\{0, \sum_{j=1}^{n} w_j s_j - b\}$
  - Seja a solução *s* = (0 1 1 1 1):
    - Soma dos benefícios: 2×0 + 2×1 + 3×1 + 4×1 + 4×1 = 13
    - Soma das capacidades: 4×0 + 5×1 + 7×1 + 9×1 + 6×1 = 27
    - $\alpha = \sum_{j=1}^{n} p_j$ = 2 + 2 + 3 + 4 + 4 = 15
    - A capacidade da mochila **não é** respeitada, então:
    - $f(s) = \sum_{j=1}^{n} p_j s_j - \alpha \times max\{0, 27 - 23\}$
    - $f(s) = 13 - 15 \times max\{0, 4\}$ = 13 – 60 = - 47

Problema da Mochila



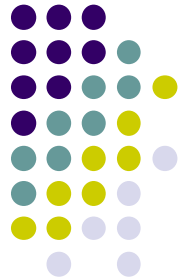
- Vizinhança de uma solução:
  - Movimento 1: Trocar os valores de dois bits diferentes entre duas posições
    - *s* = (0 **1** 1 1 **0**)
    - *s*' = (0 **0** 1 1 **1**)
  - Movimento 2: Inverter o valor de um bit
    - *s* = (0 **1** 1 1 0)
    - *s*' = (0 **0** 1 1 0)
  - Questionamento:
    - Com estes movimentos é possível explorar todo o espaço de soluções do problema?

Problema da Mochila



- No movimento de troca de bits, o número de bits iguais permanece o mesmo
  - Com ele não é possível explorar todo o espaço de soluções do problema

Problema da Mochila

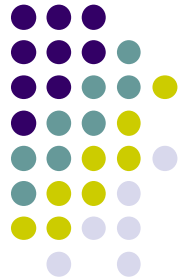


- No movimento de inversão do valor do bit é possível partir de uma solução e gerar **qualquer** outra após um número finito de operações com este movimento
  - Exemplo: partindo-se da solução inicial s = (0 1 1 1 0) é possível gerar qualquer outra solução
  - Suponhamos que a solução alvo seja s^* = (0 1 0 0 1)
  - Para alcançar s^*, basta partir da solução s e na primeira iteração, trocar s_3 por 0, na segunda iteração trocar s_4 por 0 e na terceira trocar s_5 por 1
  - s = (0 1 1 1 0) → (0 1 0 1 0) → (0 1 0 0 0) → (0 1 0 0 1) = s^*
  - Então este movimento é ESSENCIAL para explorar o espaço de soluções do problema

Problema da Mochila

- A vizinhança de inversão do valor do bit pode ser assim representada:
  - $N^{Inv}(s) = \{s' : s' \leftarrow s \oplus m_{inv}\}$
  - m_{inv} = movimento de inverter o valor de um bit
- Para a solução s = (0 1 1 1 0), a vizinhança N^{Inv} é formada pelos seguintes vizinhos:
  - s'_1 = (**1** 1 1 1 0)
  - s'_2 = (0 **0** 1 1 0)
  - s'_3 = (0 1 **0** 1 0)
  - s'_4 = (0 1 1 **0** 0)
  - s'_5 = (0 1 1 1 **1**)
- $N^{Inv}(s) = \{s'_1 , s'_2 , s'_3 , s'_4 , s'_5\}$

Problema do Caixeiro Viajante



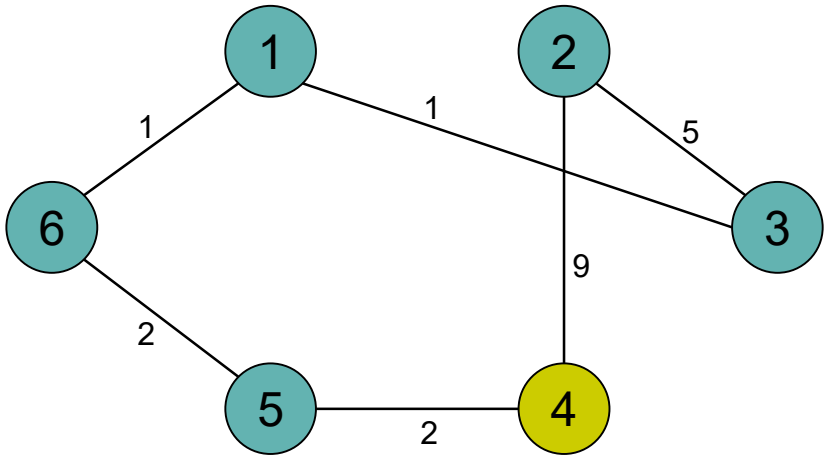
- Dado um conjunto de cidades e uma matriz de distâncias entre elas
- PCV consiste em encontrar uma rota para um Caixeiro Viajante tal que este:
  - parta de uma cidade origem
  - passe por todas as demais cidades uma única vez
  - retorne à cidade origem ao final do percurso
  - percorra a menor distância possível
- Rota conhecida como ciclo hamiltoniano

Estrutura de vizinhança para o PCV

- No PCV, um movimento *m* pode ser a troca da ordem de visita entre duas cidades
- Exemplo:
  - Dada a solução s = (6 1 3 2 4 5), são exemplos de vizinhos de s:
    - s' = (6 **4** 3 2 **1** 5)
    - s'' = (6 1 **4** 2 **3** 5)

PCV - Ex: Vizinhos de uma solução *s*

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

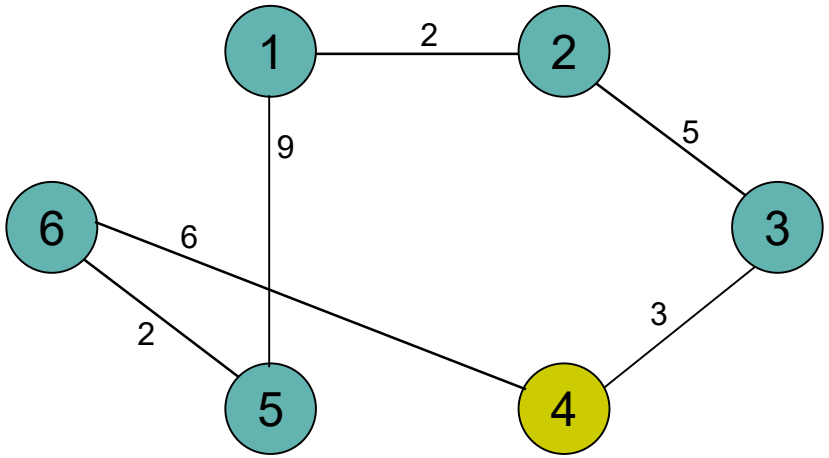


- **Distância Total *s* =** 20

$s = (6\ 1\ 3\ 2\ 4\ 5)$

PCV - Ex: Vizinhos de uma solução *s*

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



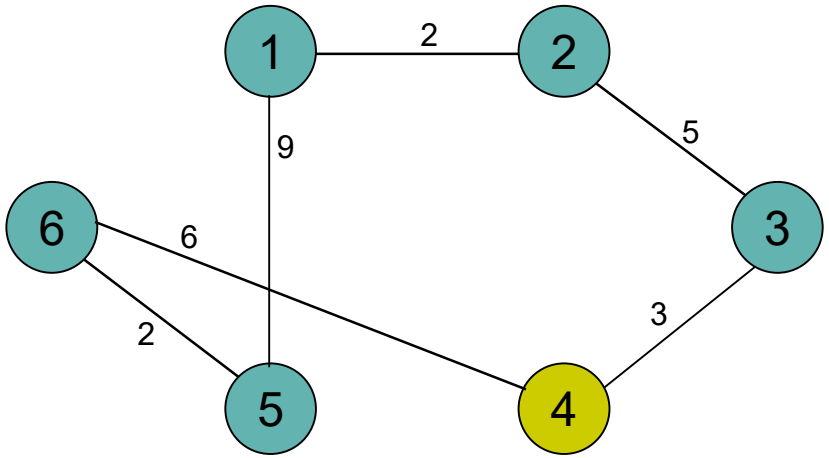
Seja *s'* obtido pela troca das cidades das posições 2 e 5, isto é, s_2=1 com s_5=4

- **Distância Total *s'* =** 27

s' = (6 **4** 3 2 **1** 5)

PCV - Ex: Vizinhos de uma solução *s*

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



Avaliação "inteligente" da solução s' = (6 **4** 3 2 **1** 5) vizinha da solução s = (6 1 3 2 4 5)

$$f(s') = f(s) - (d_{61} + d_{13} + d_{24} + d_{45}) + (d_{64} + d_{43} + d_{21} + d_{15})$$

$$f(s') = 20 - (1 + 1 + 9 + 2) + (6 + 3 + 2 + 9)$$

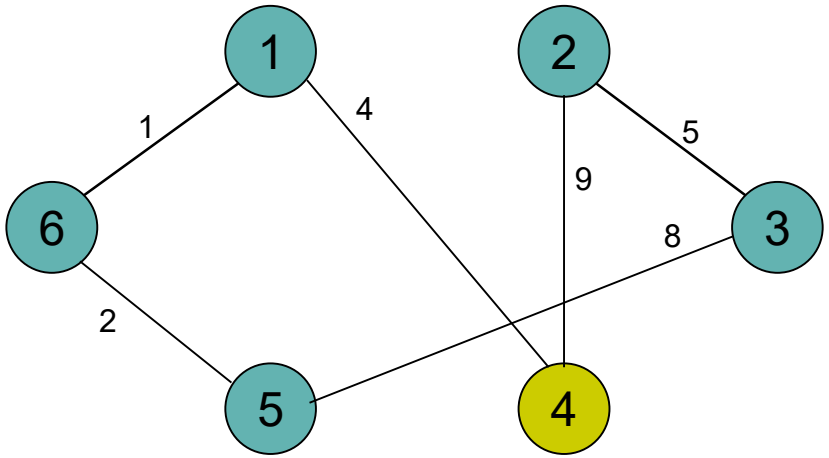
$$f(s') = 20 - (13) + (20)$$

$$f(s') = 27$$

Com esse tipo de avaliação, teremos sempre 8 operações de soma/subtração a fazer, independentemente do número de cidades!

PCV - Ex: Vizinhos de uma solução *s*

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



Seja *s'* obtido pela troca das cidades das posições 3 e 5, isto é, s_3=3 com s_5=4

- **Distância Total *s"* = 29**

s" = (6 1 **4** 2 **3** 5)

A vizinhança de troca no PCV

- Com a vizinhança de troca é possível partir de uma solução e alcançar qualquer outra do espaço de soluções do PCV?
  - Resp.: Sim!
  - Ex.: Seja *s* = (6 1 3 2 4 5) e *s** = (1 2 3 4 5 6)
- Em no máximo n-1 iterações, conseguimos partir de s e alcançar s*, aplicando somente movimentos de troca de posição
- *s* = (6 1 3 2 4 5) → (**1** 6 3 2 4 5) → (1 **2** 3 6 4 5) → (1 2 3 **4** 6 5) → (1 2 3 4 **5** 6) = *s**

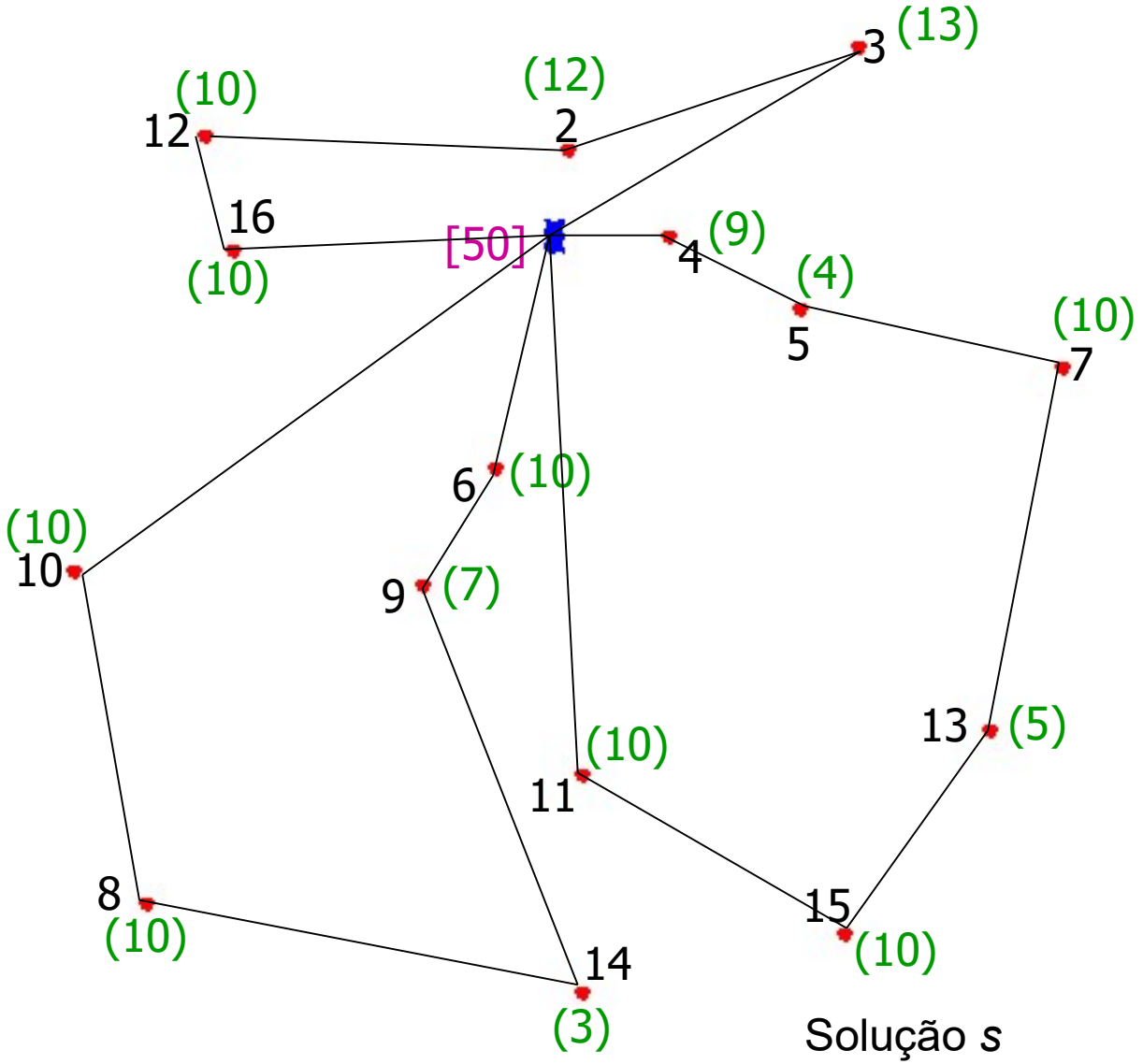
Complexidade da vizinhança de troca no PCV

- Dada uma solução $s = (s_1\ s_2\ ...\ s_{n-1}\ s_n)$ há n(n-1)/2 vizinhos de troca:

Iteração	Número de trocas
1	n – 1
2	n – 2
...	...
n - 1	1
Qtidade de vizinhos	= 1 + 2 + (n-1) = n(n-1)/2

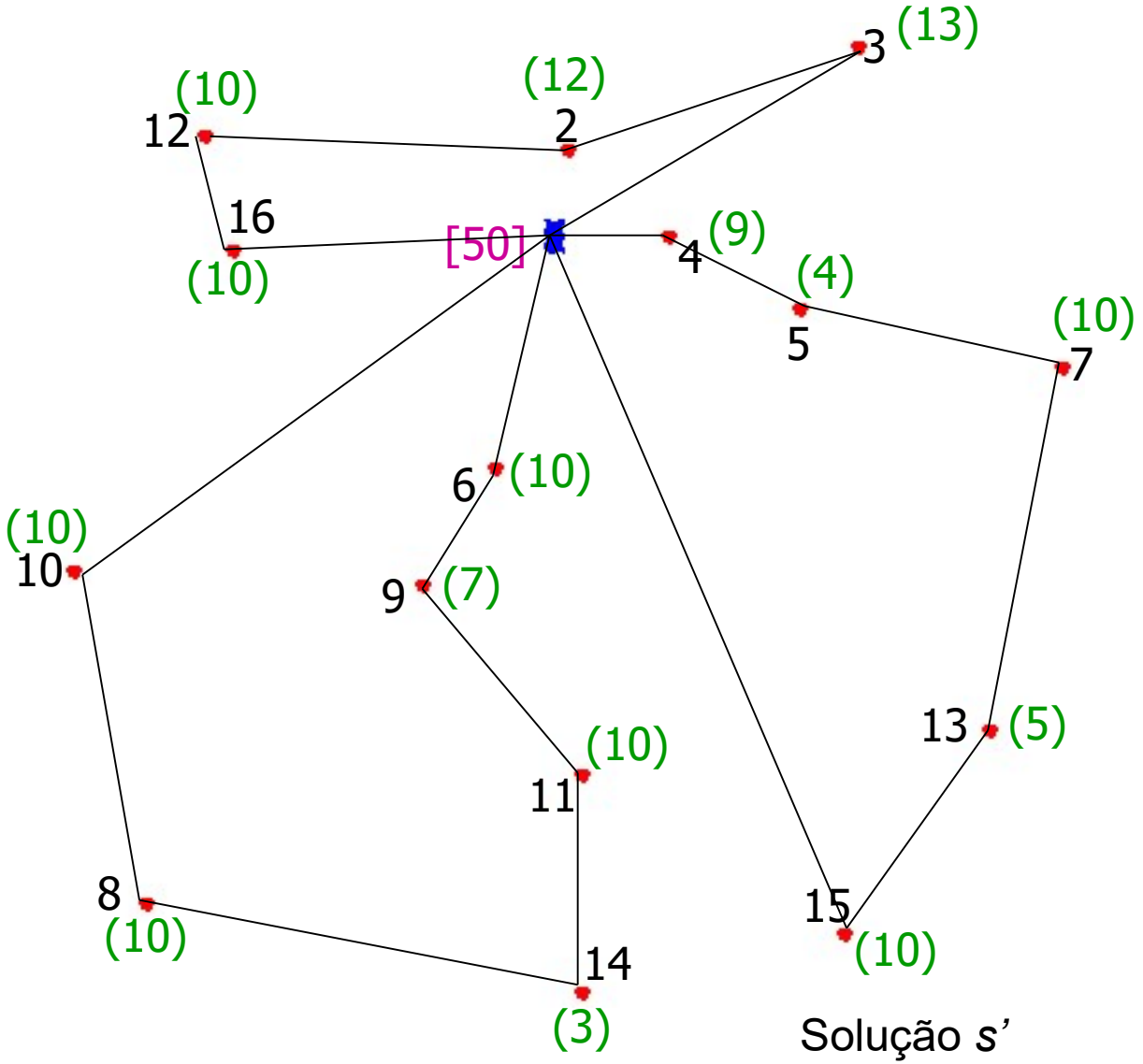
- Assim, $O(n^2)$ é a complexidade da vizinhança de troca

Exemplos de vizinhos no Problema de Roteamento de Veículos (*Vehicle Routing Problem*)



Solução *s*

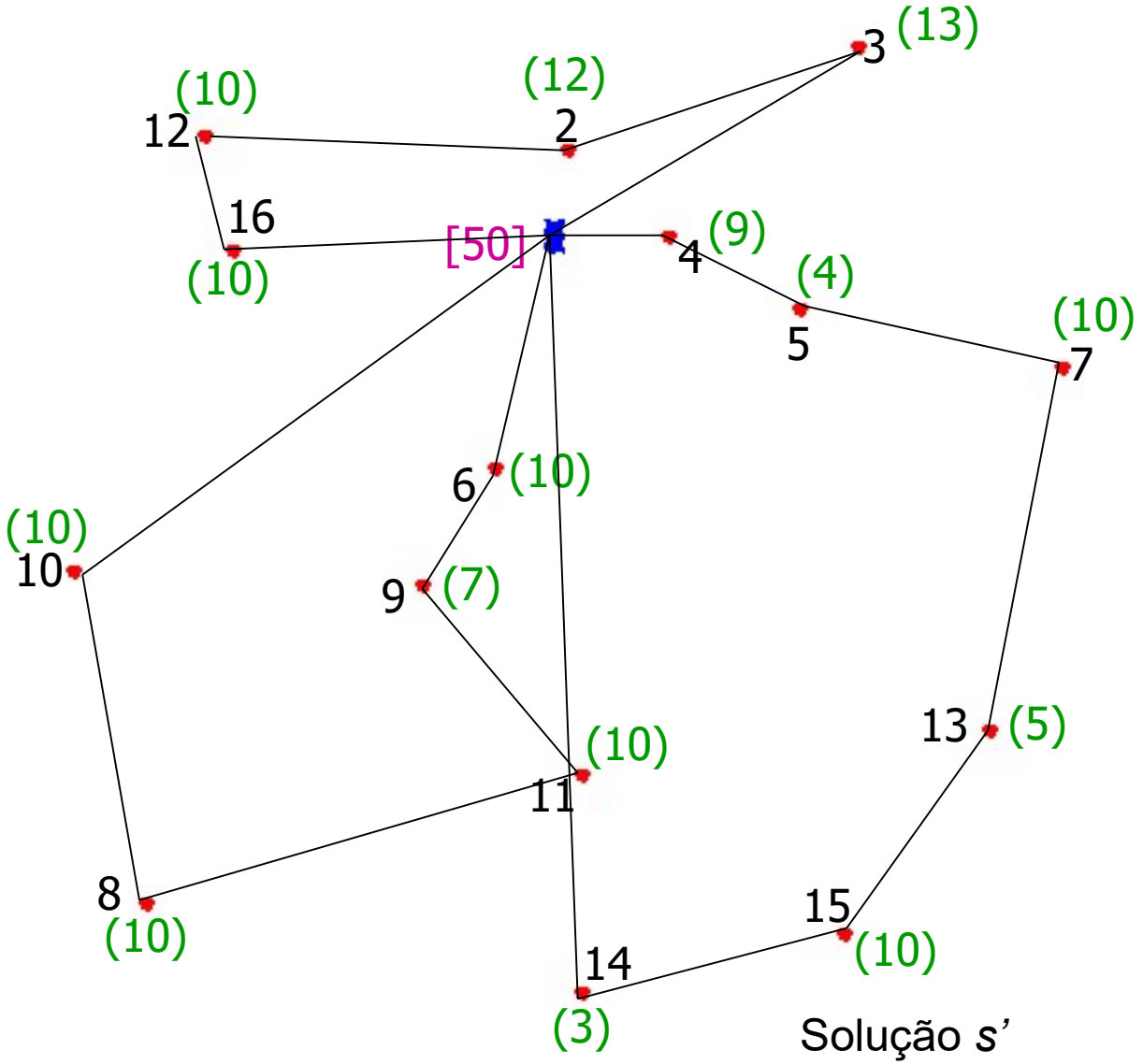
Exemplos de vizinhos no Problema de Roteamento de Veículos (*Vehicle Routing Problem*)



Solução *s'*

s' obtida pela **realocação** de um cliente de uma rota para outra rota (realocação inter-rota)

Exemplos de vizinhos no Problema de Roteamento de Veículos (*Vehicle Routing Problem*)



Solução *s'*

s' obtida pela **troca** de clientes entre rotas (movimento inter-rotas)

(Clientes 11 e 14 mudam de rota)

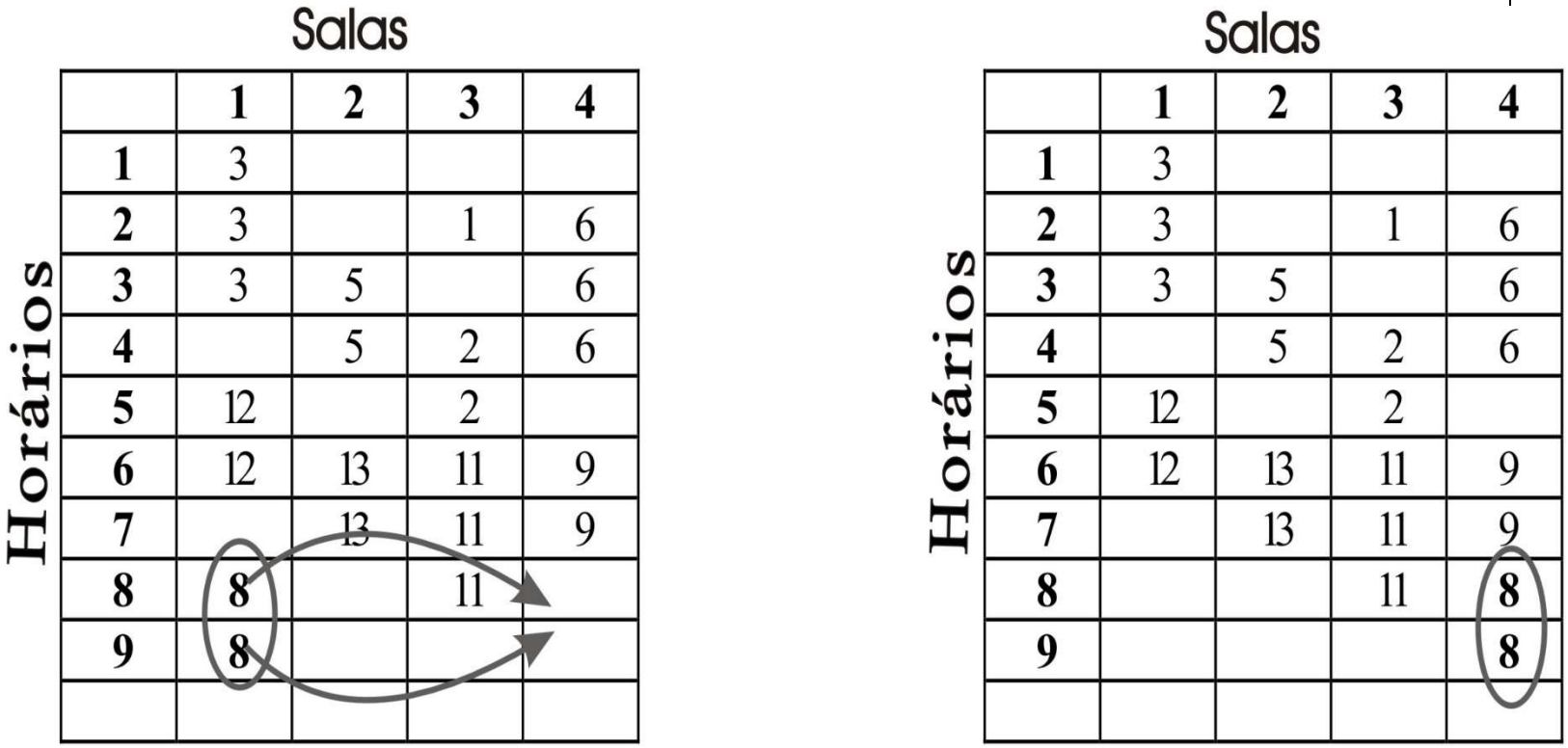
Problema de Alocação de Salas (*Classroom Assignment Problem*)

- Diz respeito à designação de salas para as aulas de uma instituição de ensino
- O horário das aulas é previamente conhecido
- O PAS é um subproblema do Problema de Programação de Horários (*timetabling*)
- Pode ser tratado de forma isolada ou de forma integrada à programação de horários

Problema de Alocação de Salas (*Classroom Assignment Problem*)

- Restrições:
  - Não pode haver sobreposição de turmas;
  - As salas têm que comportar as turmas etc.
- Objetivos:
  - Manter as aulas de uma mesma turma em uma mesma sala ao longo da semana;
  - Minimizar o fluxo de alunos mudando de sala após uma aula;
  - Sempre que possível, alocar a uma mesma sala alunos de um mesmo curso e período etc.

Problema de Alocação de Salas (*Classroom Assignment Problem*)

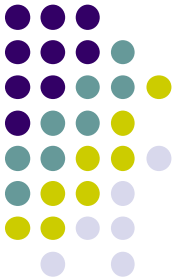


Horários \ Salas	1	2	3	4
1	3			
2	3		1	6
3	3	5		6
4		5	2	6
5	12		2	
6	12	13	11	9
7		13	11	9
8	8		11	
9	8			

Horários \ Salas	1	2	3	4
1	3			
2	3		1	6
3	3	5		6
4		5	2	6
5	12		2	
6	12	13	11	9
7		13	11	9
8			11	8
9				8

Movimento de Realocação

Problema de Alocação de Salas (*Classroom Assignment Problem*)



Salas

Horários	1	2	3	4
1	3			
2	3		1	6
3	3	5		6
4		5	2	6
5	12		2	
6	12	13	11	9
7		13	11	9
8	8		11	
9	8			

Salas

Horários	1	2	3	4
1	3			
2	3	6	1	
3	3	6		5
4		6	2	5
5	12		2	
6	12	13	11	9
7		13	11	9
8	8		11	
9	8			

Movimento de Troca

Problema de Alocação de Salas (*Classroom Assignment Problem*)

Algumas possíveis estruturas de vizinhança:

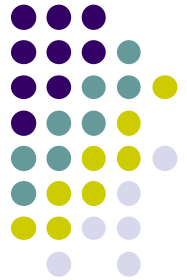
- $N^1(s) = \{s' \mid s' \leftarrow s \oplus$ *movimento de realocação*$\}$
- $N^2(s) = \{s' \mid s' \leftarrow s \oplus$ *movimento de troca*$\}$
- $N(s) = \{s' \mid s' \leftarrow s \oplus$ *mov. de realocação ou troca*$\}$

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Representação de uma Solução

	CARGA	***Cam_1***	***Cam_2***	***...***	***Cam_V***
F_1	$(Car_2, 0)$	5	X	...	4
F_2	$(D, 0)$	0	0	...	0
F_3	$(Car_1, 1)$	2	4	...	1
...	...	...	...	...	...
F_F	$(Car_5, 1)$	0	9	...	3

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

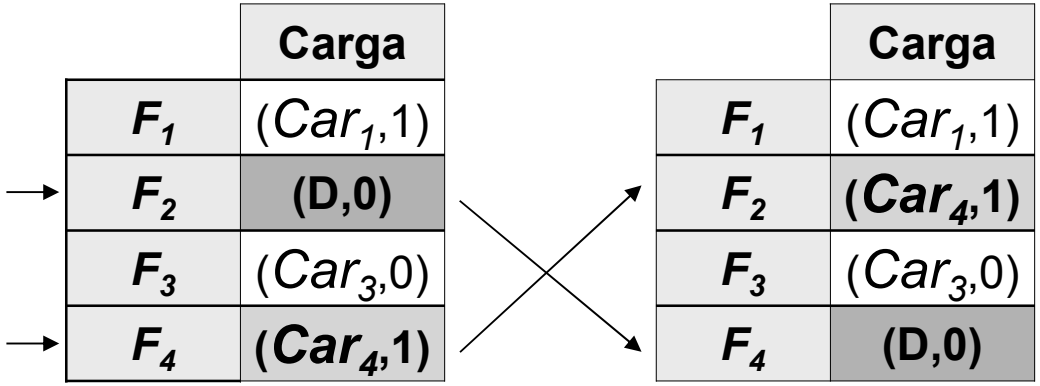


- Seis tipos de movimentos para explorar o espaço de soluções:
  - Movimento Carga
  - Movimento Número de Viagens
  - Movimento Realocar Viagem de um Caminhão
  - Movimento Realocar Viagem a uma Frente
  - Movimento Operação Caminhão
  - Movimento Operação Frente
- Cada movimento define um tipo de vizinhança

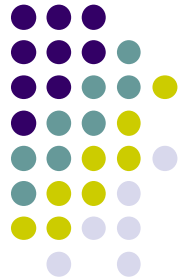
Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Vizinhança N^{CG}

Movimento Carga - $N^{CG}(s)$



Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões



Vizinhança N^{OF}

Movimento Operação Frente - $N^{OF}(s)$

Desativar operação de uma carregadeira alocada a uma frente

	Carga
F_1	**(Car_1,1)**
F_2	(D,0)
F_3	(Car_3,0)
F_4	(Car_4,1)

→

	Carga
F_1	**(Car_1,0)**
F_2	(D,0)
F_3	(Car_3,0)
F_4	(Car_4,1)

Ativar operação de uma carregadeira alocada a uma frente

	Carga
F_1	(Car_1,1)
F_2	(D,0)
F_3	**(Car_3,0)**
F_4	(Car_4,1)

→

	Carga
F_1	(Car_1,0)
F_2	(D,0)
F_3	**(Car_3,1)**
F_4	(Car_4,1)

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Vizinhança N^{NV}

Movimento Número de Viagens - $N^{NV}(s)$

Decréscimo no número de viagens de um caminhão a uma frente

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	**6**	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	4	3

-1 →

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	**5**	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	4	3

Acréscimo no número de viagens de um caminhão a uma frente

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	4	**3**

+1 →

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	4	**4**

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Vizinhança N^{VC}

Movimento Realocar Viagem de um Caminhão - $N^{VC}(s)$

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	**6**	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	**2**	3

1

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	**5**	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	**3**	3

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Vizinhança N^{VF}

Movimento Realocar Viagem a uma Frente - $N^{VF}(s)$

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
→ F_4	(Car_4,1)	**4**	3

1 →

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	**3**	4

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Vizinhança N^{OC}

Movimento Operação Caminhão - $N^{OC}(s)$

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
→ F_4	(Car_4,1)	**4**	3

↑

→

	Carga	Cam_1	Cam_2
F_1	(Car_1,1)	6	X
F_2	(D,0)	0	0
F_3	(Car_3,0)	0	0
F_4	(Car_4,1)	**0**	3

↑

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Função de avaliação de uma solução *s*:

- Feita por uma função que penaliza o não atendimento às restrições e objetivos

- Restrições (Requisitos essenciais):
  - Produção da mina dentro dos limites de especificação;
  - Parâmetros de controle respeitam os limites de qualidade especificados;
  - Relação estéril/minério dentro dos limites de especificação;
  - Taxa de utilização dos caminhões inferior ao máximo possível;
  - Produção dos equipamentos de carga respeita as capacidades de produção especificadas.

- Objetivos (Requisitos não-essenciais):
  - Atendimento às metas de produção da mina
  - Atendimento às metas de qualidade dos parâmetros de controle
  - Atendimento à relação estéril/minério desejada
  - Taxa de utilização de caminhões igual à meta de utilização
  - Utilização do menor número possível de caminhões.

$$f(s) = f^{p}(s) + \sum_{j \in S} f_j^{q}(s) + f^{r}(s) + \sum_{l \in V} f_l^{u}(s) + \sum_{i \in F} f_i^{c}(s) + f^{n}(s)$$

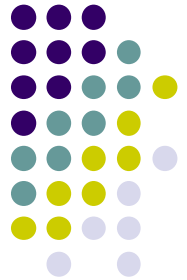
Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Avaliação da solução *s* quanto à produção:

$$f^{p}(s) = \theta^{p} \times \left| Pr - P \right|$$

P : Produção de minério (t/h);
Pr : Meta de produção de minério (t/h);
θ^{p} : Peso associado à avaliação da produção.

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

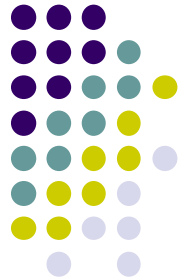


Avaliação da solução *s* quanto à qualidade:

$$f_j^q(s) = \theta_j^q \times \Delta_j^q \times \left| Qr_j - Q_j \right| \quad \forall j \in S$$

Q_j : Quantidade encontrada na mistura para o parâmetro j (t/h);
Qr_j : Quantidade recomendada para o parâmetro j na mistura (t/h);
θ_j^q : Peso associado à avaliação da qualidade do parâmetro j;
Δ_j^q : Multiplicador associado ao parâmetro j.

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões



Avaliação da solução *s* quanto à relação estéril / minério:

$$f^r(s) = \theta^r \times \left| Rr - R \right|$$

R : Produção de estéril na mistura (t/h);

R_r : Meta de Produção de estéril na mistura (t/h);

θ_r : Peso associado à avaliação da relação estéril/minério

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Avaliação da solução *s* quanto à taxa de utilização dos caminhões:

$$f_l^u(s) = \theta_l^u \times \left| Ur_l - U_l \right| \quad \forall l \in V$$

U_l : Carga transportada pelo caminhão l (t/h);
Ur_l : Meta de carga transportada pelo caminhão l (t/h);
θ_l^u : Peso associado à avaliação da utilização do caminhão l

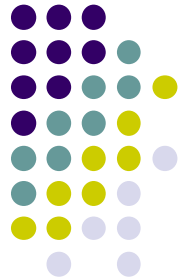
Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões

Avaliação da solução *s* quanto à produção dos equipamentos de carga:

$$f_i^c(s) = \theta_k{}^c \times \left| Cu_k - x_i \right| \quad \forall i \in F$$

x_i : *Ritmo de lavra da frente i (t/h);*
k : *Equipamento de carga que está operando na frente i;*
Cu^k : *Produção máxima do equipamento de carga k alocado à frente i (t/h);*
θ_k^c : *Peso associado à avaliação da produção do equipamento de carga k alocada à frente i*

Planejamento Operacional de Lavra com Alocação Dinâmica de Caminhões



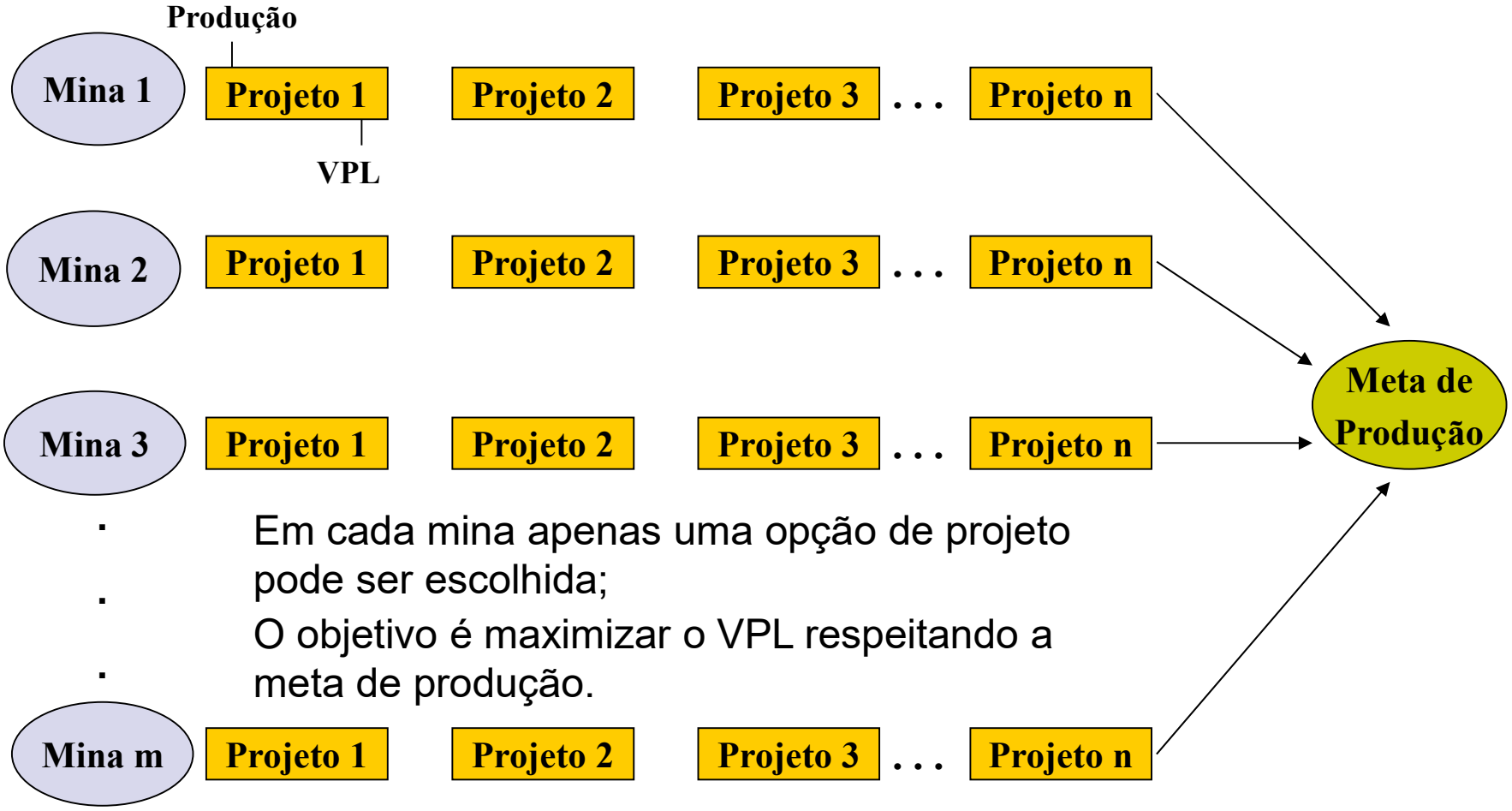
Avaliação da solução *s* quanto ao número de caminhões utilizados:

$$f^n(s) = \beta^n \times \sum_{i \in F} \lceil TU_l \rceil$$

TU_l : Taxa de utilização do caminhão l , em %;

β_n : Peso associado à avaliação do número total de caminhões utilizados

Problema de Seleção de Projetos Mineiros Concorrentes



- Em cada mina apenas uma opção de projeto pode ser escolhida;
- O objetivo é maximizar o VPL respeitando a meta de produção.

Modelagem Heurística para o Problema de Seleção de Projetos

Representação de uma solução *s*:

- Vetor de m posições, sendo m o número de minas
- Cada célula s_j representa o projeto implementado na mina j

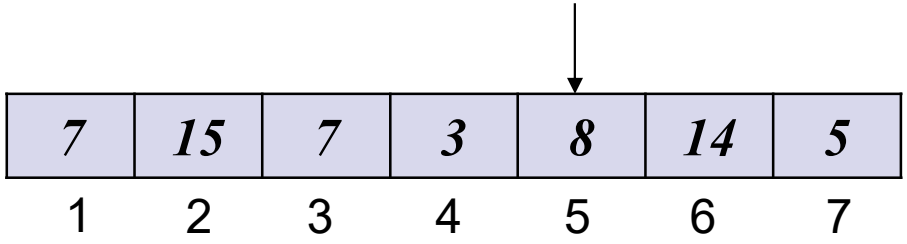
7	15	7	3	8	14	5
1	2	3	4	5	6	7

*Obs.: Com esta representação, a restrição de que em cada mina um projeto tem que ser implementado é **automaticamente** satisfeita

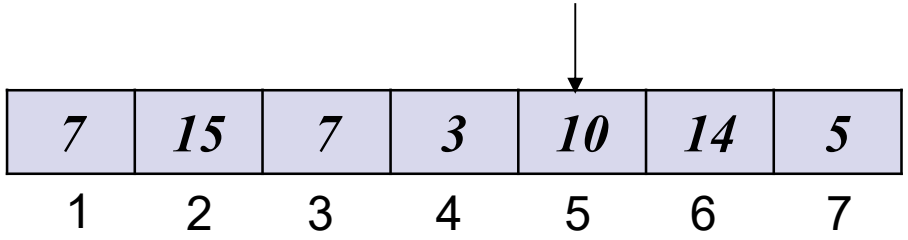
Estrutura de Vizinhança para o Problema de Seleção de Projetos

Exploração do espaço de soluções por meio do Movimento Substituição de Projeto

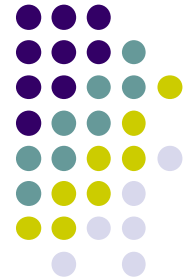
Solução *s*:



Exemplo de vizinho *s'*: gerado a partir de s, substituindo-se o projeto implementado em uma mina por outro projeto



Avaliação da Solução heurística do Problema da Seleção de Projetos



Feita por função que procura maximizar o VPL e penalizar a produção inferior ou superior à meta

$$f(s) = \sum_{i \in Minas} \sum_{j \in Opcoes} VPL_{i,S_i} - Pfalta \times \max\{0, M - \sum_{i \in Minas} prod_{i,S_i}\} - Pexc \times \max\{0, \sum_{i \in Minas} prod_{i,S_i} - M\}$$

em que:

Minas = conjunto de minas;

Opcoes = conjunto das opções de projeto em cada mina;

VPL_{i,S_i} = Valor Presente Líquido referente à opção j da mina i;

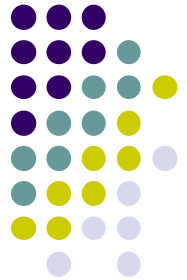
$prod_{i,S_i}$ = Produção referente à opção j da mina i;

Pfalta = Penalidade por produção inferior à meta;

Pexc = Penalidade por produção superior à meta;

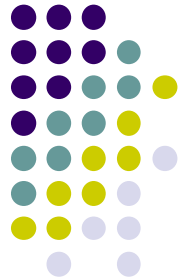
M = Meta de produção especificada.

Heurísticas de refinamento (Princípio de funcionamento)



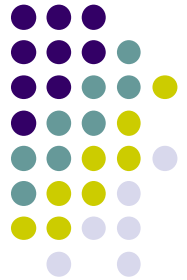
- Partir de uma solução inicial qualquer
- Caminhar, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada, tentando melhorar a solução construída

Método da descida/subida (*Descent/Uphill Method*)



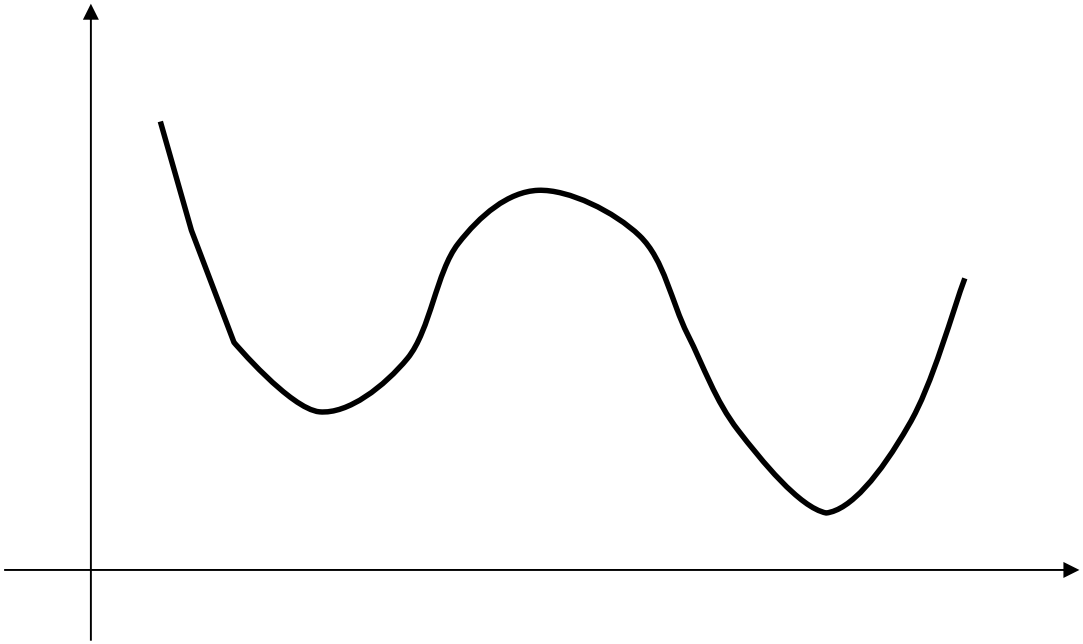
- Estratégias de busca:
  - Completa: *Best Improvement*
  - Primeira melhora: *First Improvement*
  - Randômica: *Random Improvement*
- Cada estratégia de busca define um método diferente!
  - Descent Method with best improvement strategy (BI)
  - Descent Method with first improvement strategy (FI)
  - Random Descent (RD)

Método da descida/subida (*Descent/Uphill Method*) com estratégia Best Improvement

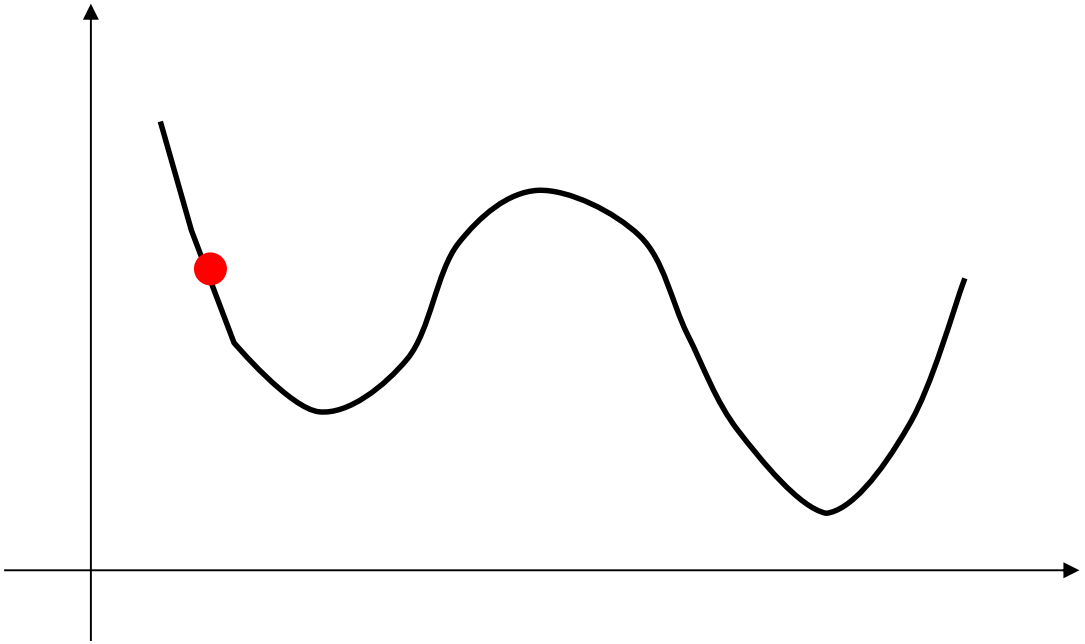


- Parte-se de uma solução inicial qualquer
- A cada passo analisam-se todos os possíveis vizinhos da solução corrente *s*
- Se o melhor vizinho *s'* for melhor que a solução corrente *s*, então move-se para esse vizinho
  - Em problemas de minimização, um vizinho *s'* é melhor que *s* quando f(*s'*) < f(*s*))
  - Neste caso, *s'* passa a ser a nova solução corrente e repete-se o procedimento de análise da vizinhança
- Caso contrário, o método é encerrado e retorna-se um ótimo local em relação à vizinhança usada

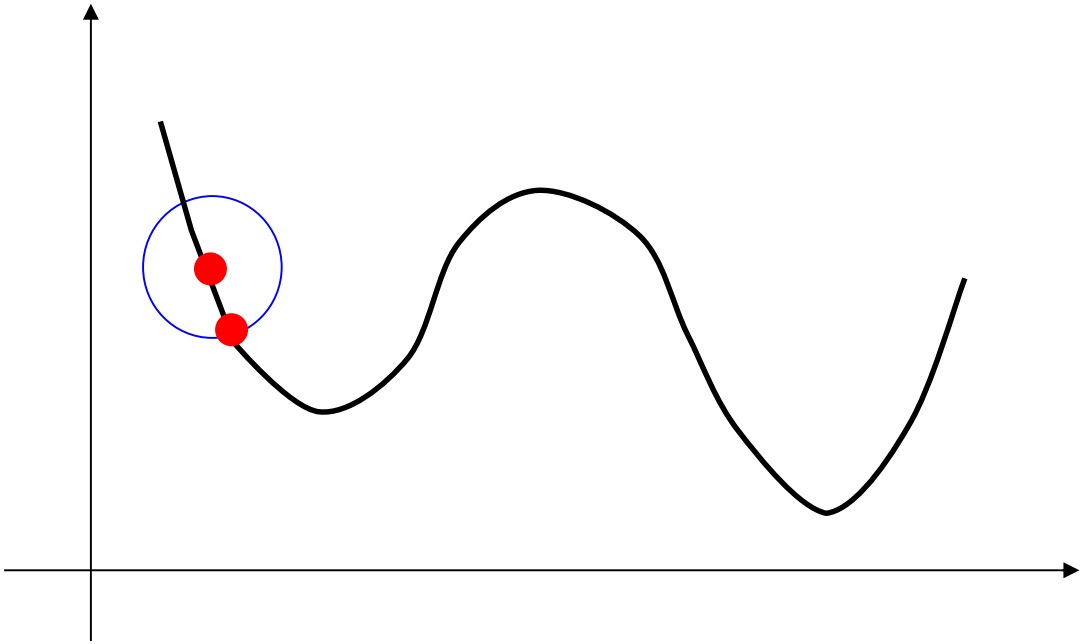
Funcionamento das Heurísticas de Descida



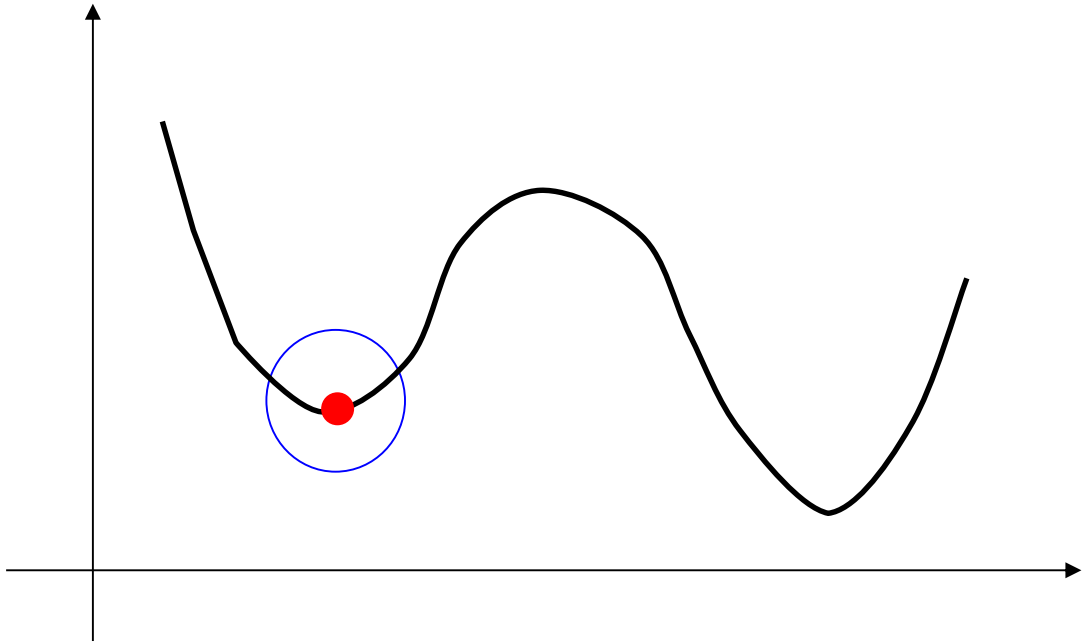
Funcionamento das Heurísticas de Descida



Funcionamento das Heurísticas de Descida



Funcionamento das Heurísticas de Descida

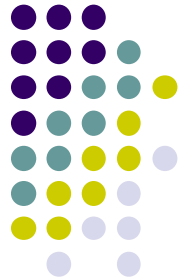


Problema: Fica-se preso no primeiro ótimo local

Pseudocódigo do método da Descida com a estratégia Best Improvement

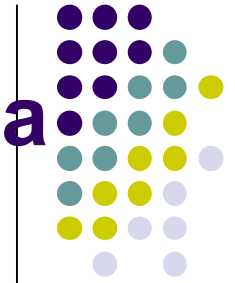
```
procedimento Descida(f(.), N(.), s);
1   V = {s' ∈ N(s) | f(s') < f(s)};
2   enquanto (|V| > 0) faça
3       Selecione s' ∈ V, onde s' = arg min{f(s') | s' ∈ V};
4       s ← s' ;
5       V = {s' ∈ N(s) | f(s') < f(s)};
6   fim-enquanto;
7   Retorne s;
fim Descida;
```

Método da descida/subida com estratégia First Improvement



- Variante do Método de Descida/Subida
- Evita a pesquisa exaustiva pelo melhor vizinho
- Consiste em interromper a exploração da vizinhança quando um vizinho melhor é encontrado
- Desta forma, apenas no pior caso, toda a vizinhança é explorada
- A solução final é um ótimo local com respeito à vizinhança considerada

Método da descida/subida randômica (Random Descent/Uphill Method)



- Variante do Método de Descida/Subida
- Evita a pesquisa exaustiva pelo melhor vizinho
- Consiste em escolher aleatoriamente um vizinho e o aceitar somente se ele for de melhora
- Se o vizinho escolhido não for de melhora, a solução corrente permanece inalterada e outro vizinho é aleatoriamente gerado
- O procedimento é interrompido após um certo número fixo de iterações sem melhora no valor da melhor solução obtida até então
- A solução final **não** é necessariamente um ótimo local

Método de Descida Randômica (*Random Descent Method*)

```
procedimento DescidaRandomica(f(.), N(.), IterMax, s);
1   Iter ← 0;      {Contador de iterações sem melhora }
2   enquanto (Iter < IterMax) faça
3       Iter ← Iter + 1;
4       Selecione aleatoriamente s' ∈ N(s);
5       se (f(s') < f(s)) então
6           Iter ← 0;
7           s ← s' ;
8       fim-se;
9   fim-enquanto;
10  Retorne s;
fim DescidaRandomica;
```