Tópicos de Análise de Algoritmos

Algoritmos de aproximação

Uma introdução

Sec 11.1 e 11.2 do KT

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
  um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
  uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
  um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
  uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problema de minimização: Dada uma instância I viável,
encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é mínimo.

Problema de maximização: Dada uma instância I viável,
encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é máximo.

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problema de minimização: Dada uma instância I viável,
encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é mínimo.

Problema de maximização: Dada uma instância I viável,
encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é máximo.

$\mathrm{opt}(I)$: valor ótimo (valor de uma solução ótima)

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Em algoritmos de aproximação, abrimos mão de (1):

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Em algoritmos de aproximação, abrimos mão de (1):

buscamos *boas* soluções que
possam ser obtidas eficientemente.

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

Para problemas de maximização,
a desigualdade é invertida e $\alpha < 1$.

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

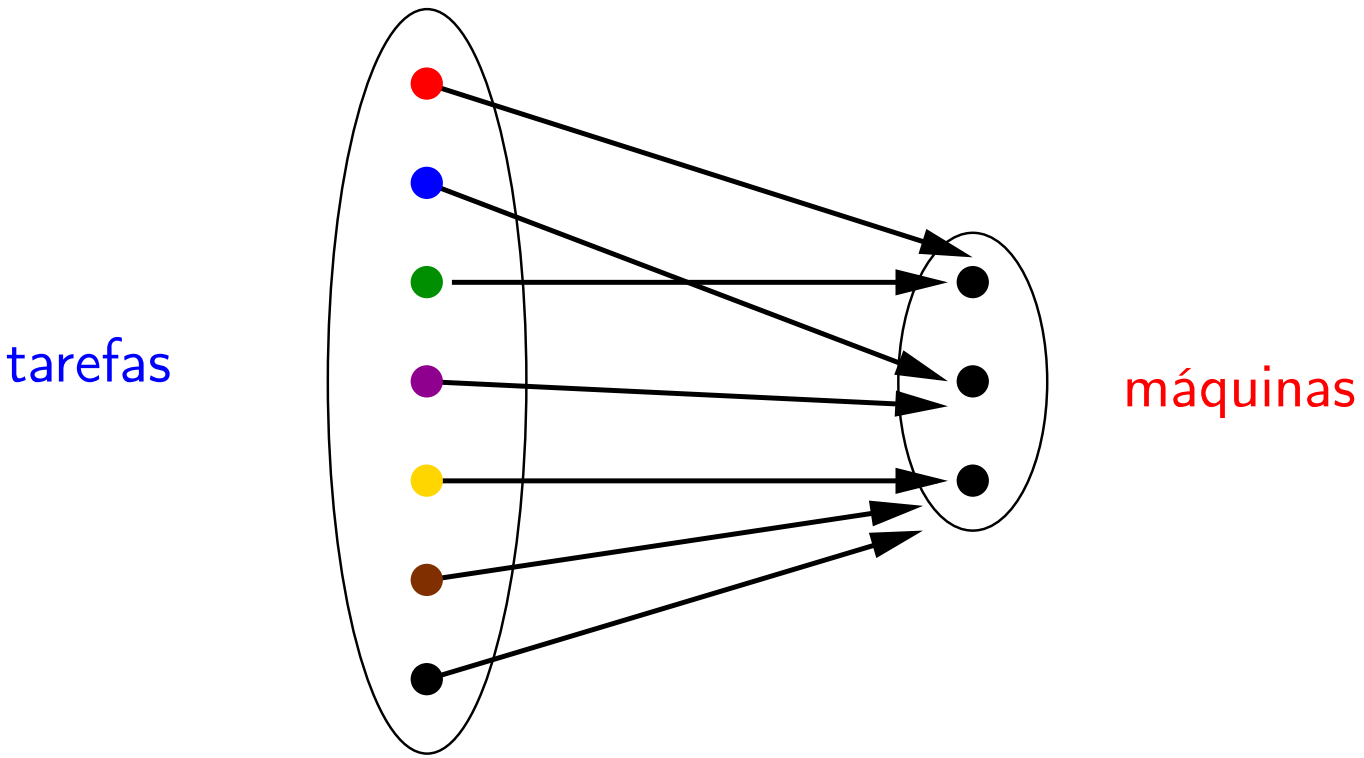
Para problemas de maximização,
a desigualdade é invertida e $\alpha < 1$.

Objetivo: α tão perto de 1 quanto possível

Escalonamento em máquinas idênticas

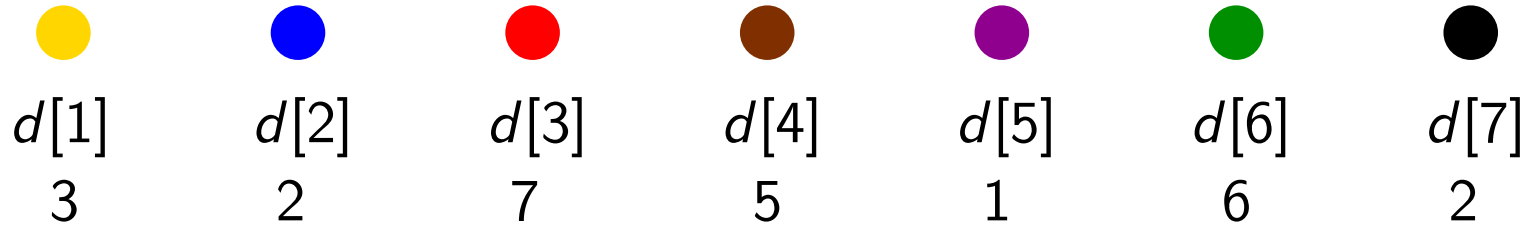
Dados: q máquinas
n tarefas
duração $d[i]$ da tarefa i $\quad (i = 1, \ldots, n)$

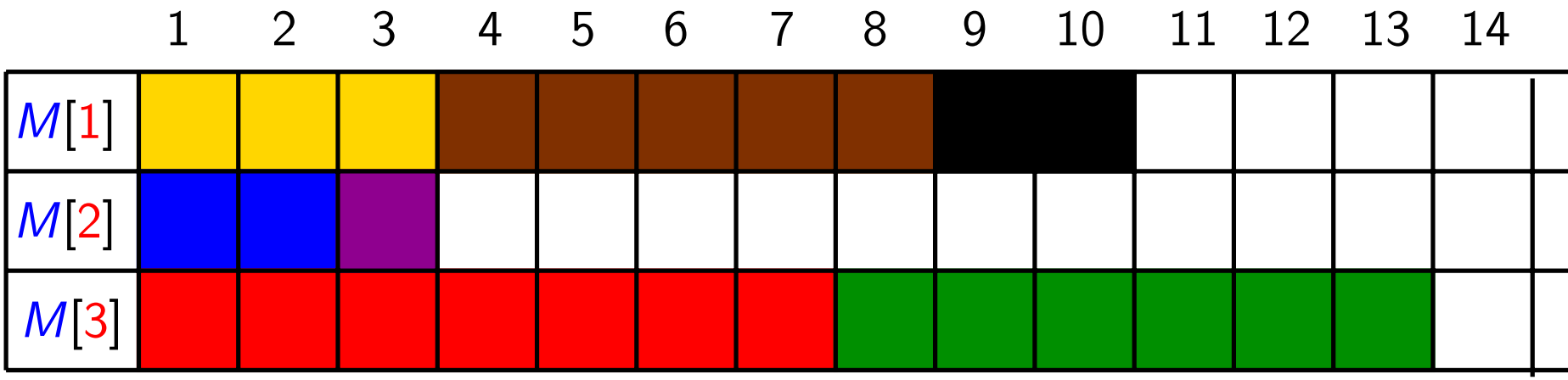
escalonamento: partição $\{M[1], \ldots, M[q]\}$ de $\{1, \ldots, n\}$



Exemplo 1

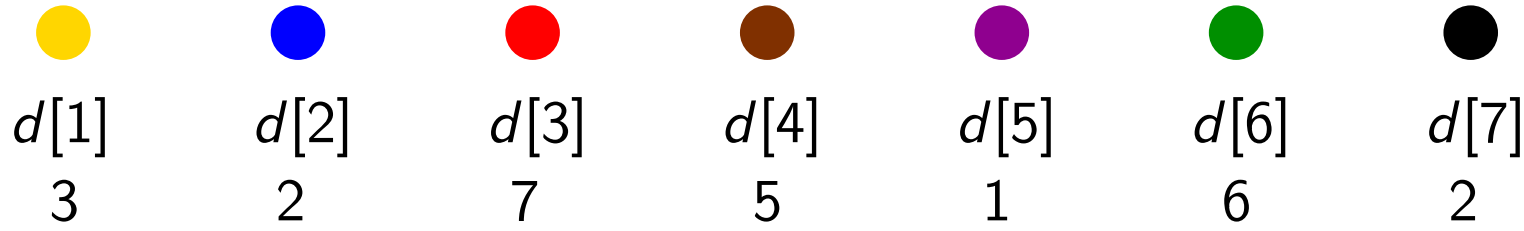
$q = 3 \quad n = 7$

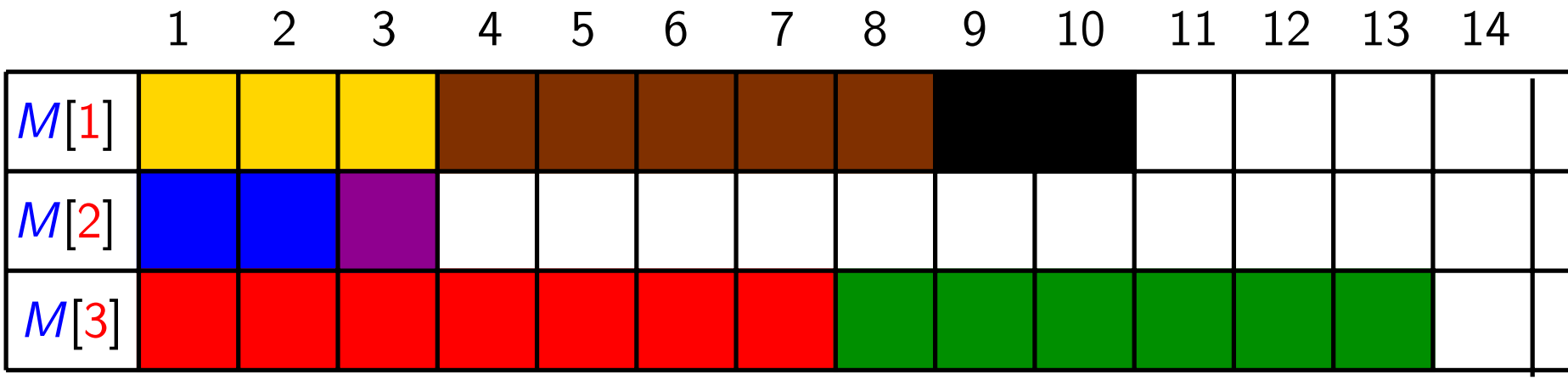




Exemplo 1

$q = 3 \quad n = 7$

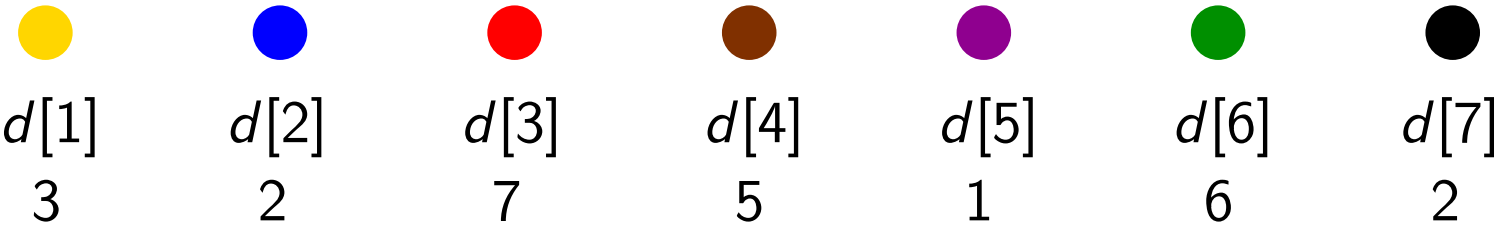


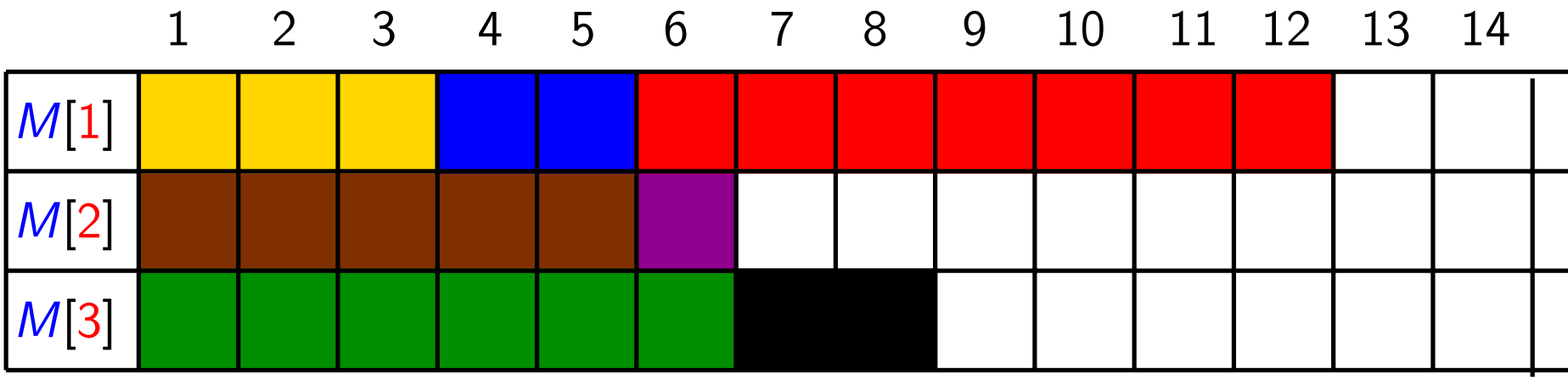


$\{\{1,4,7\},\{2,5\},\{3,6\}\} \Rightarrow$ Tempo de conclusão $= 13$

Exemplo 2

$q = 3 \quad n = 7$

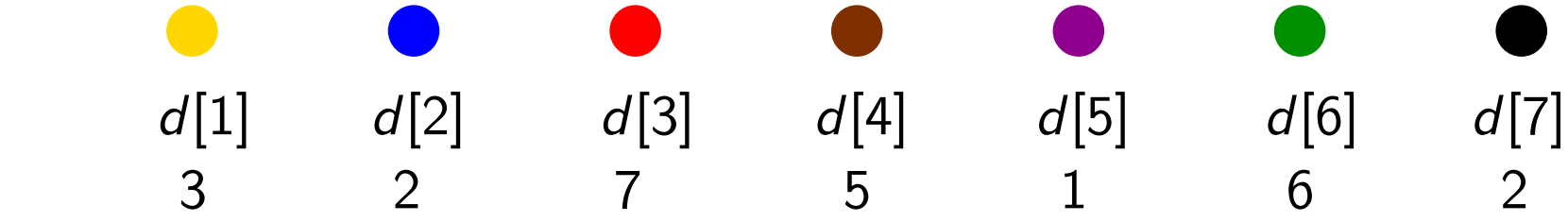


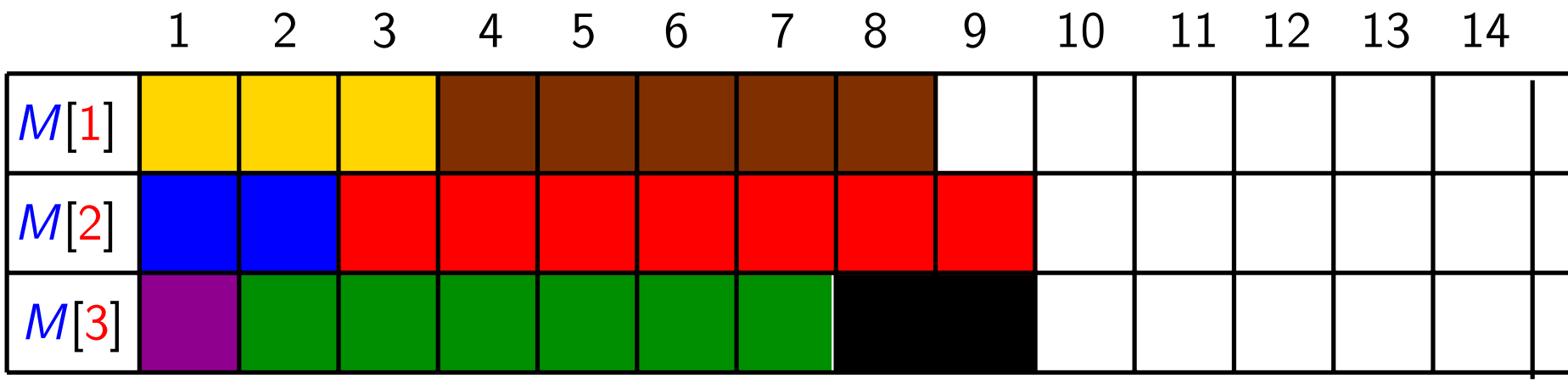


$\{\{1,2,3\},\{4,5\},\{6,7\}\} \Rightarrow$ Tempo de conclusão $= 12$

Problema

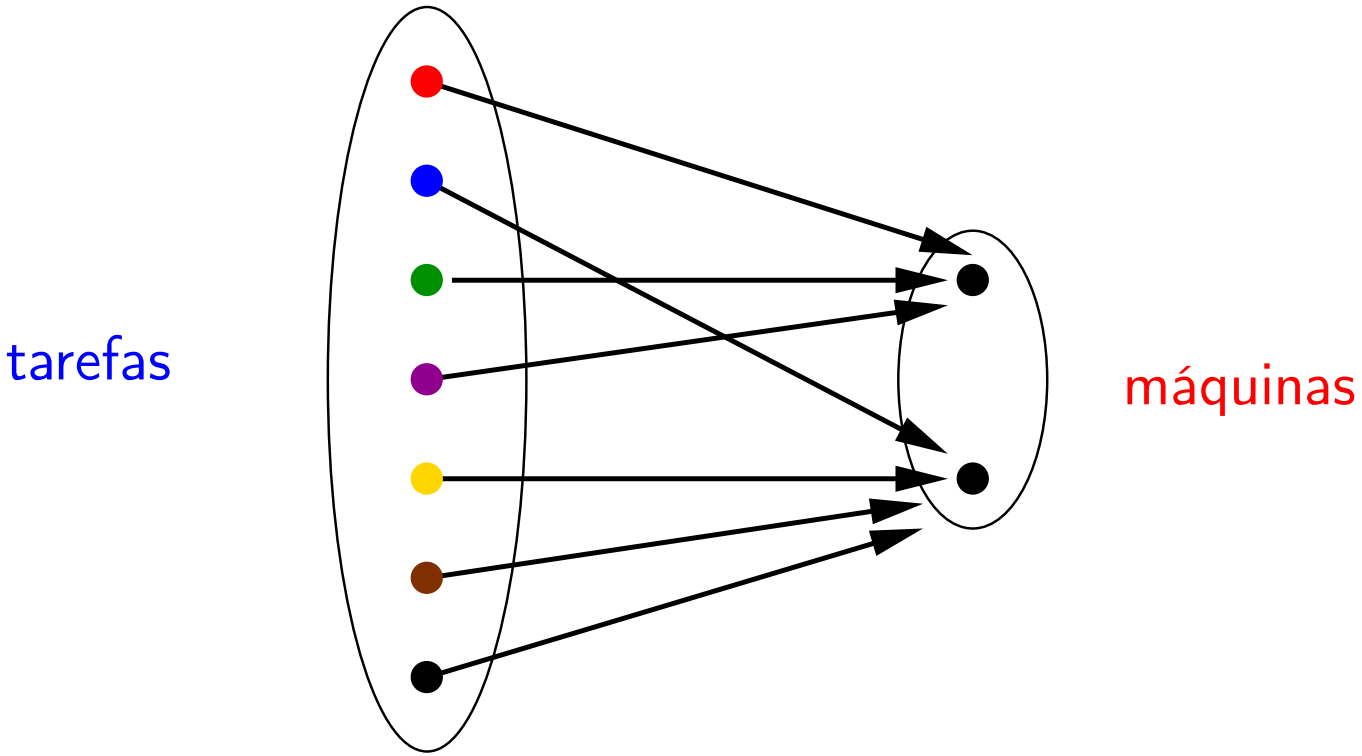
Encontrar um escalonamento com tempo de conclusão mínimo.





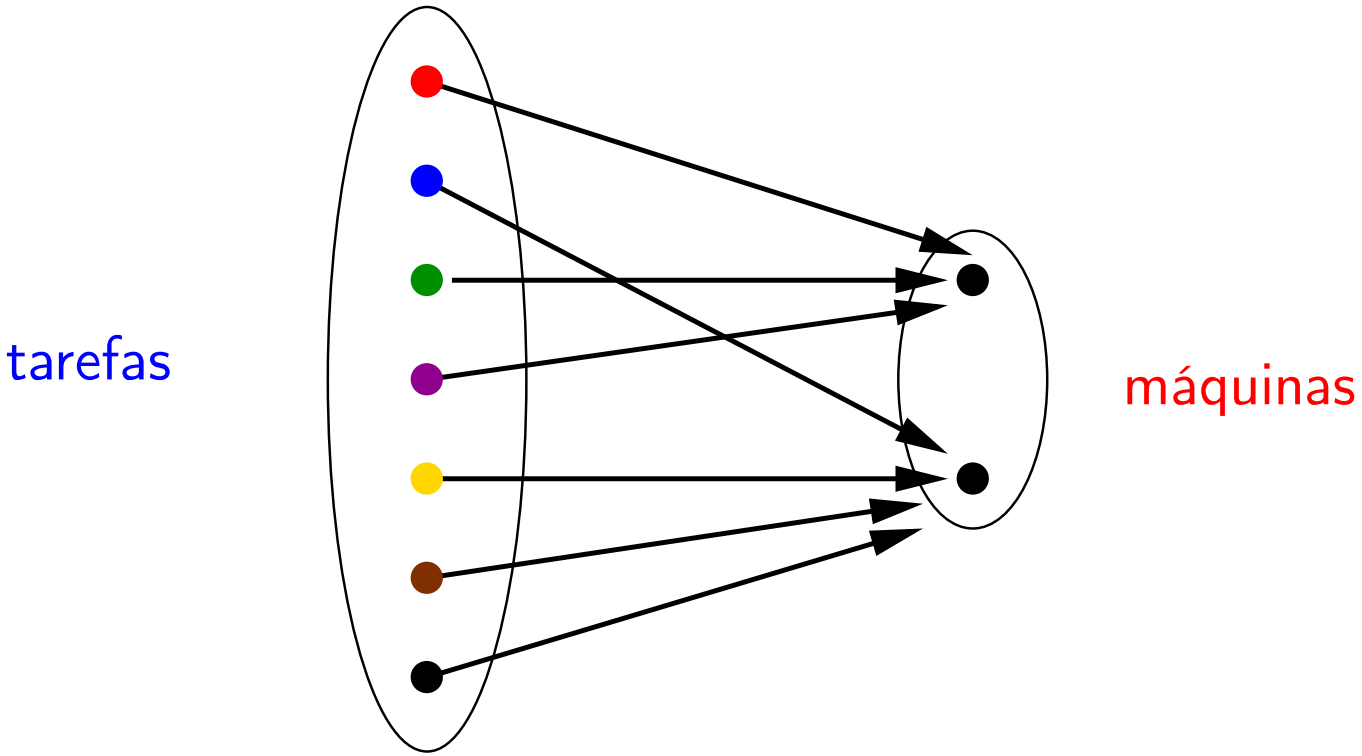
$\{\{1,4\},\{2,3\},\{5,6,7\}\} \Rightarrow$ Tempo de conclusão $= 9$

NP-difícil mesmo para $q = 2$



Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, n\}$ e escolhe melhor

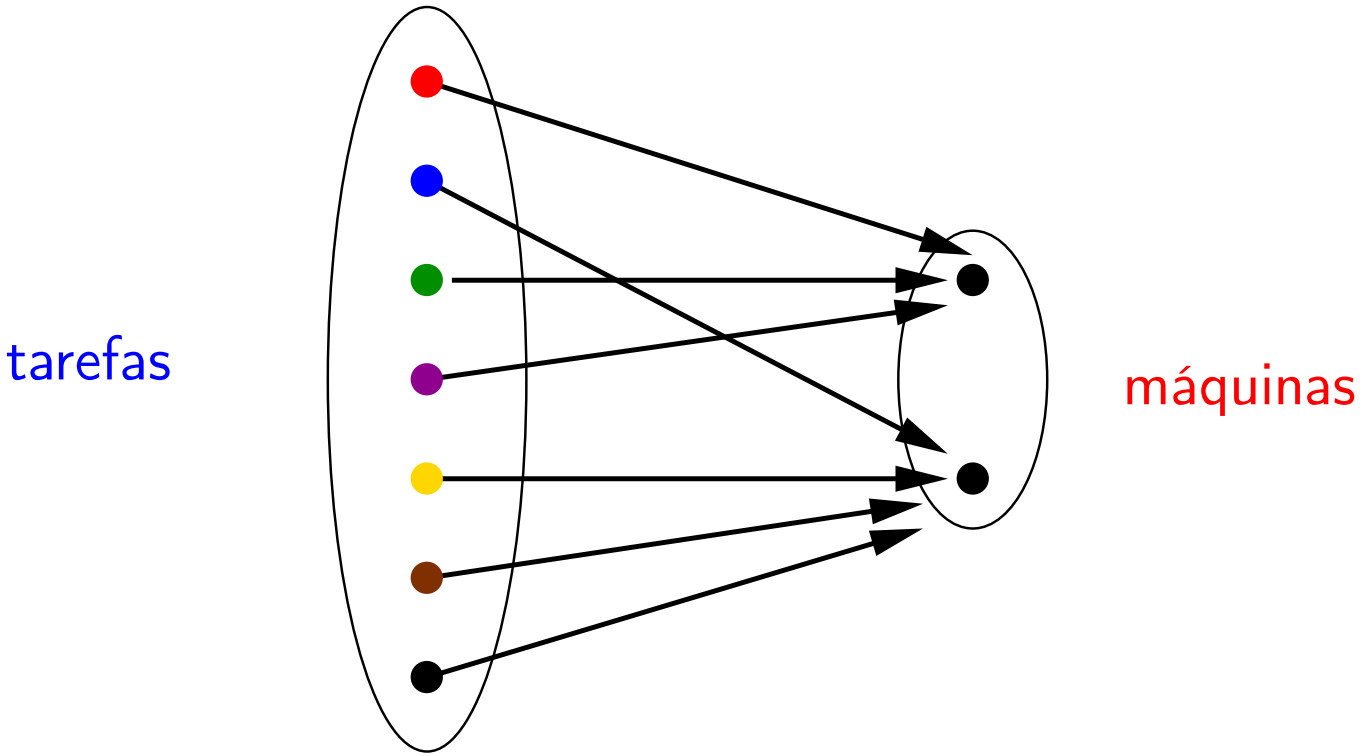
NP-difícil mesmo para $q = 2$



Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, n\}$ e escolhe melhor

2^n subconjuntos $\Rightarrow$ exponencial

NP-difícil mesmo para $q = 2$



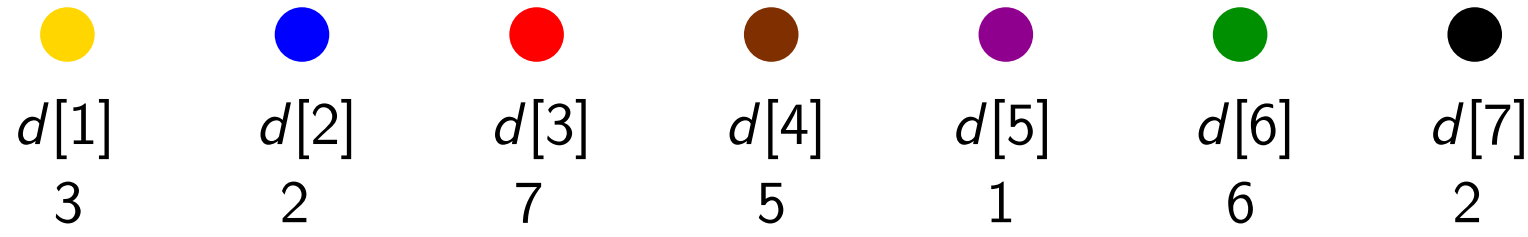
Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, n\}$ e escolhe melhor

2^n subconjuntos $\Rightarrow$ exponencial

NP-difícil $\Rightarrow$ é improvável que exista algoritmo polinomial que resolva o problema (se existir, P = NP)

Algoritmo de Graham

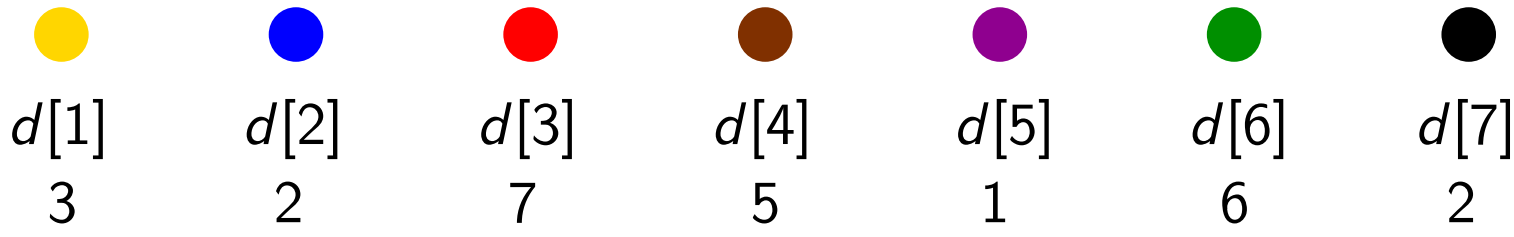
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

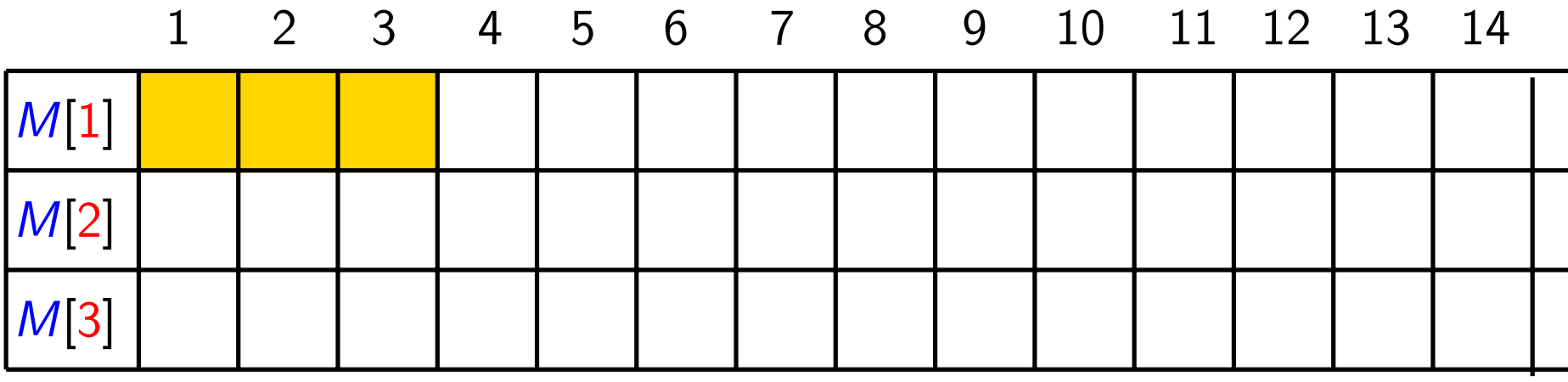


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$M[1]$														
$M[2]$														
$M[3]$														

Algoritmo de Graham

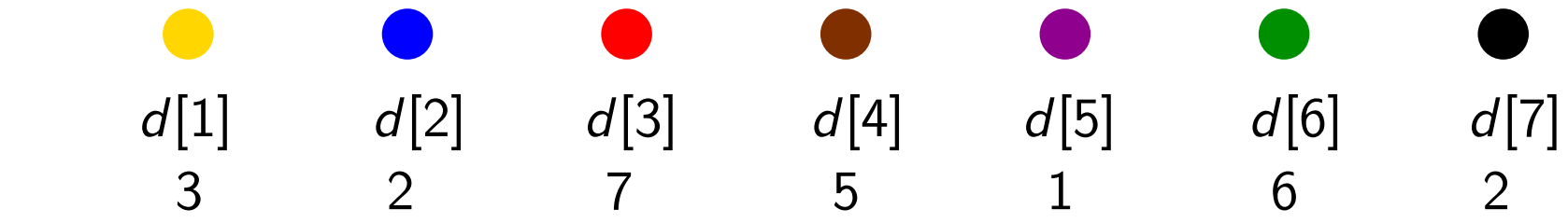
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

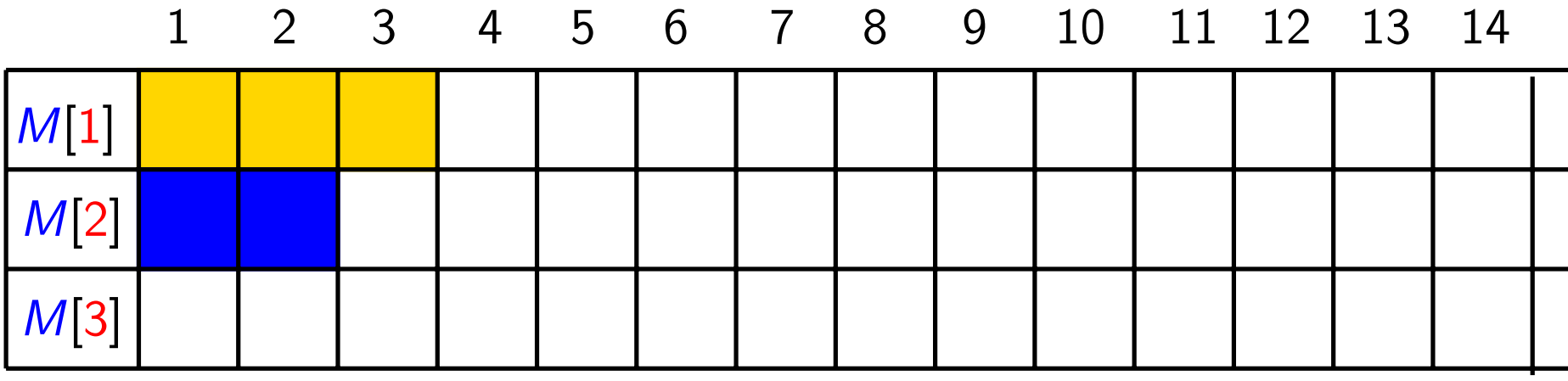




Algoritmo de Graham

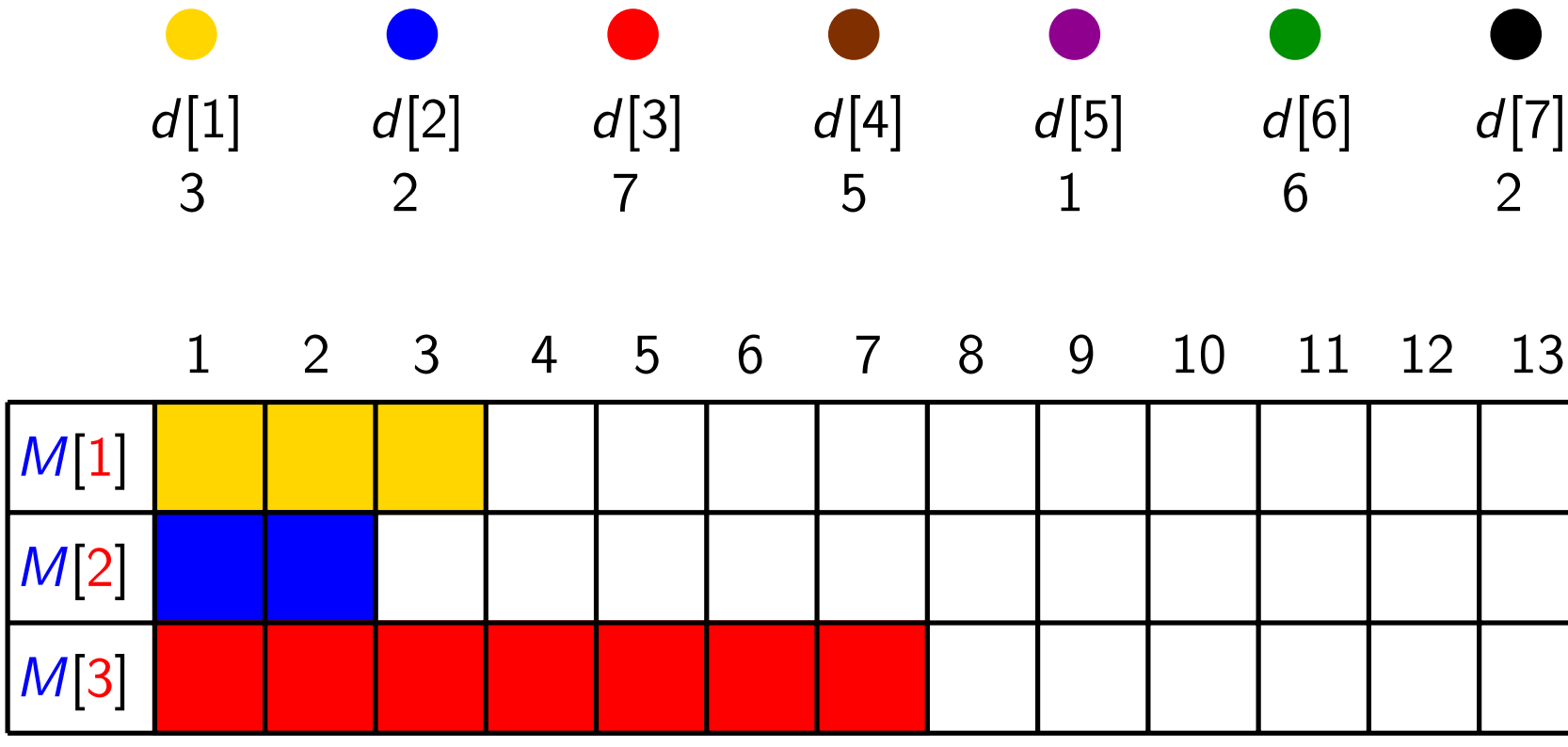
Atribui, uma a uma, cada tarefa à máquina menos ocupada.





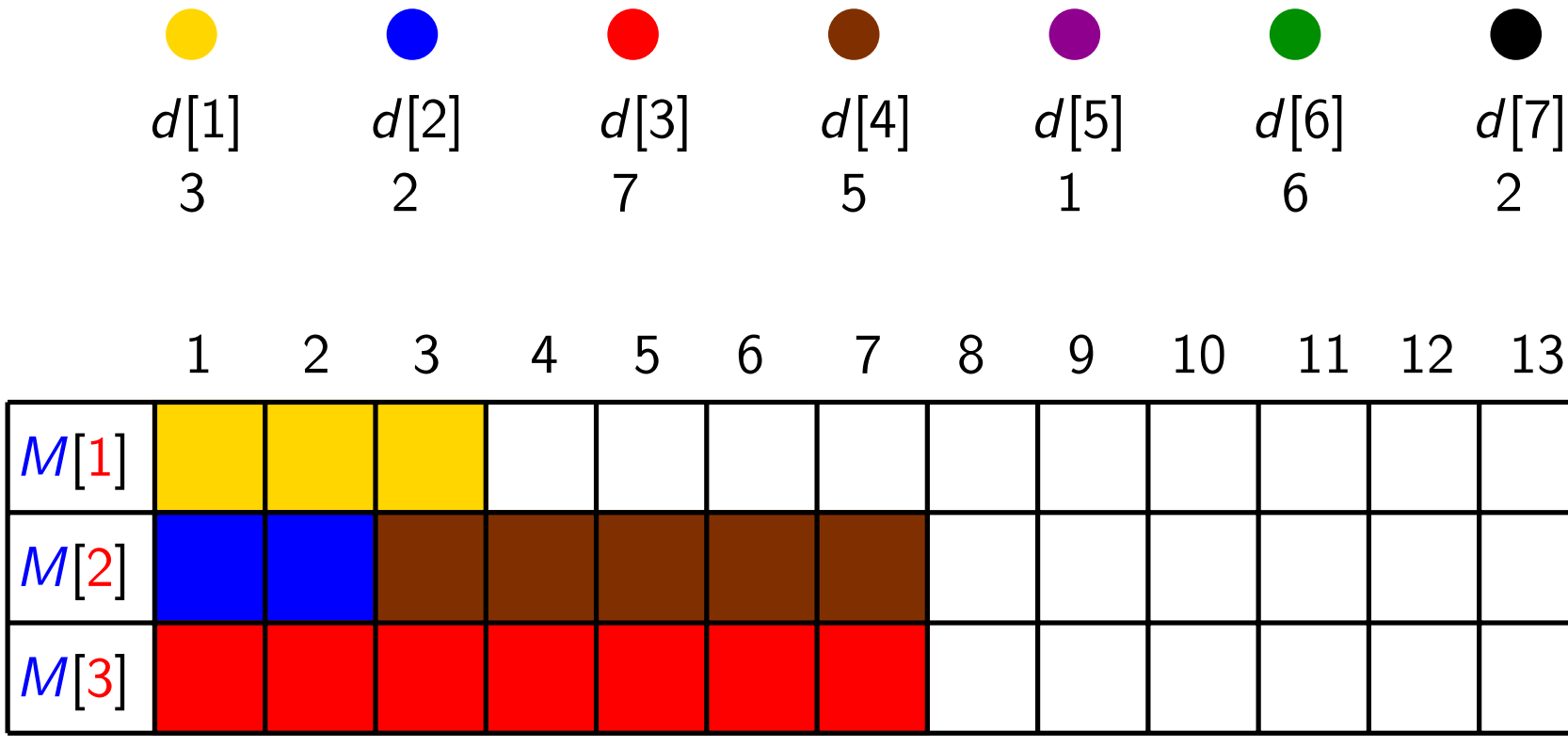
Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



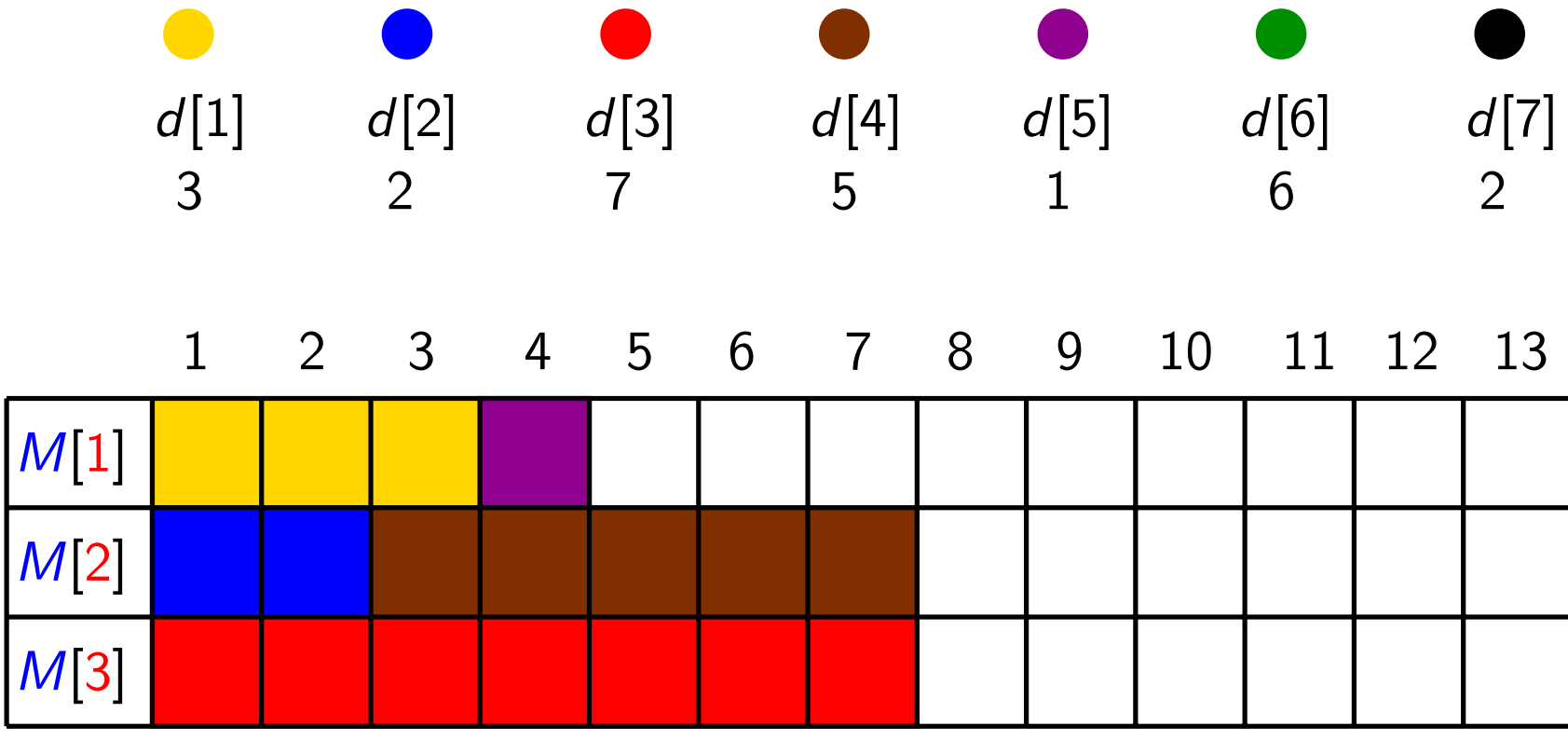
Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



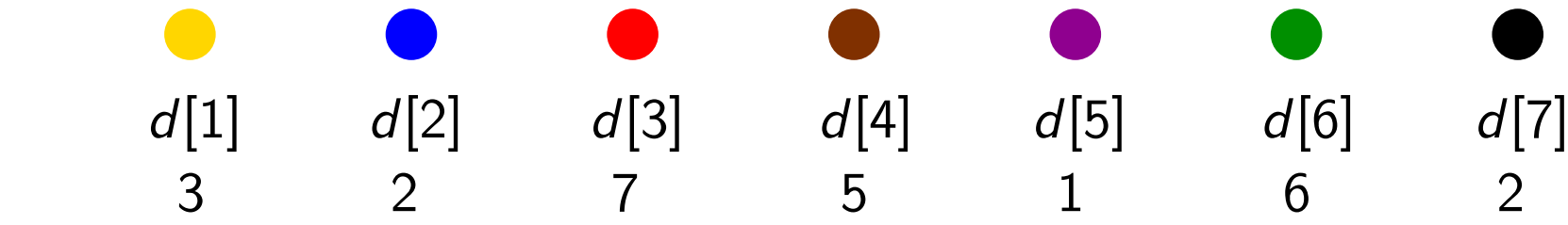
Algoritmo de Graham

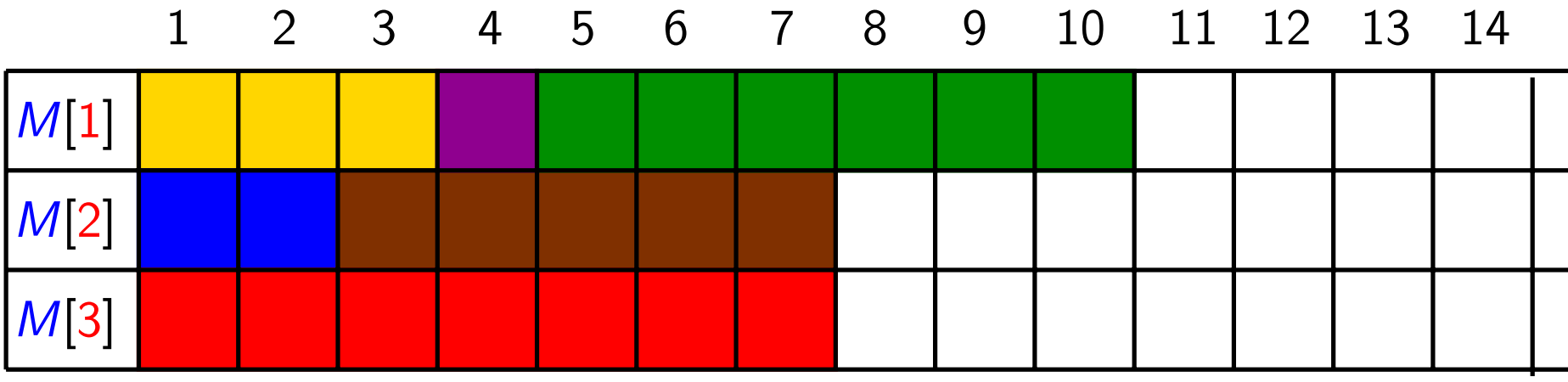
Atribui, uma a uma, cada tarefa à máquina menos ocupada.



Algoritmo de Graham

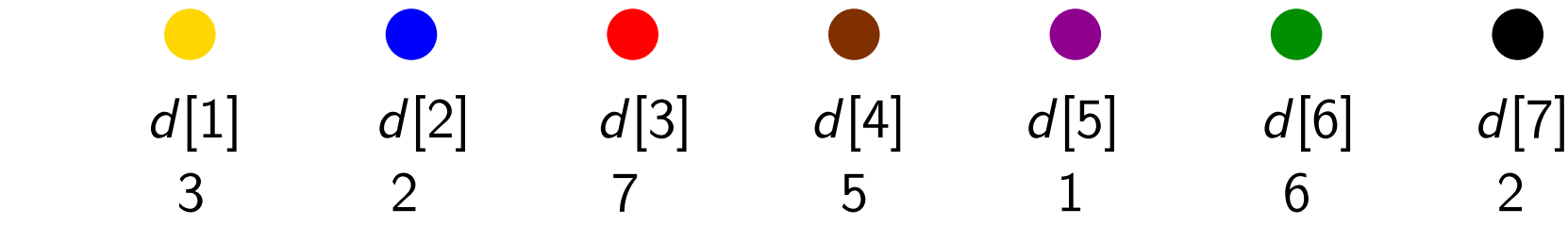
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

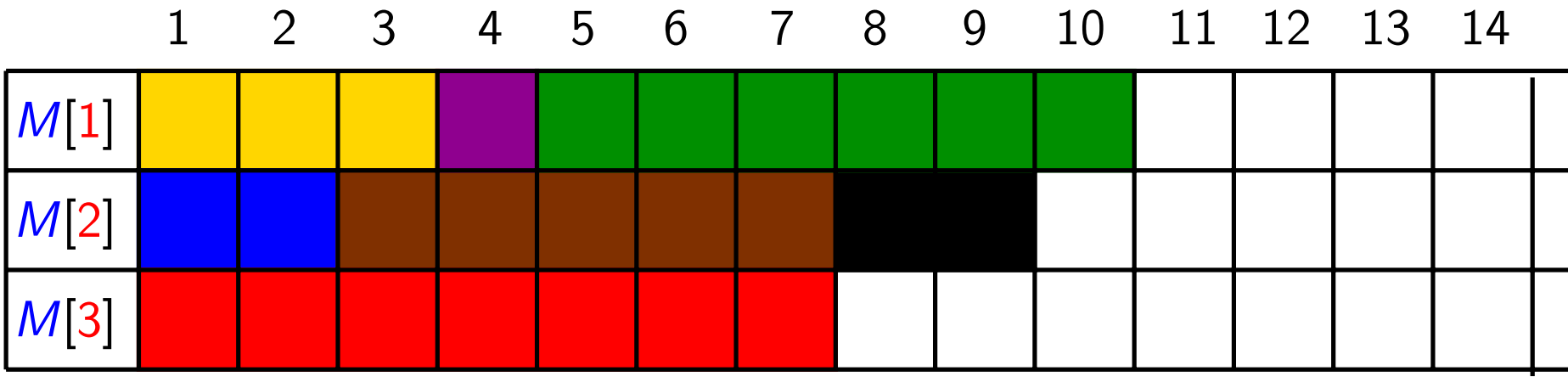




Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.





Escalonamento-Graham

Recebe números inteiros positivos q e n e um vetor $d[1\,..\,n]$ e devolve um escalomento de $\{1, \ldots, n\}$ em q máquinas.

ESCALONAMENTO-GRAHAM (q, n, d)

```
1    para j ← 1 até q faça
2            M[j] ← ∅
3            T[j] ← 0
```

Escalonamento-Graham

Recebe números inteiros positivos q e n e um vetor $d[1\,..\,n]$ e devolve um escalomento de $\{1, \ldots, n\}$ em q máquinas.

ESCALONAMENTO-GRAHAM (q, n, d)

```
1    para j ← 1 até q faça
2          M[j] ← ∅
3          T[j] ← 0
4    para i ← 1 até n faça
5          seja k tal que T[k] é mínimo
```

Escalonamento-Graham

Recebe números inteiros positivos q e n e um vetor $d[1\,..\,n]$ e devolve um escalomento de $\{1, \ldots, n\}$ em q máquinas.

ESCALONAMENTO-GRAHAM (q, n, d)

```
1   para j ← 1 até q faça
2       M[j] ← ∅
3       T[j] ← 0
4   para i ← 1 até n faça
5       seja k tal que T[k] é mínimo
6       M[k] ← M[k] ∪ {i}
7       T[k] ← T[k] + d[i]
8   devolva {M[1], ..., M[q]}
```

Delimitações para opt

opt = menor tempo de conclusão de um escalonamento

Delimitações para opt

opt = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

$$\mathrm{opt} \geq \max\{d[1], d[2], \ldots, d[n]\}$$

Delimitações para opt

opt = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

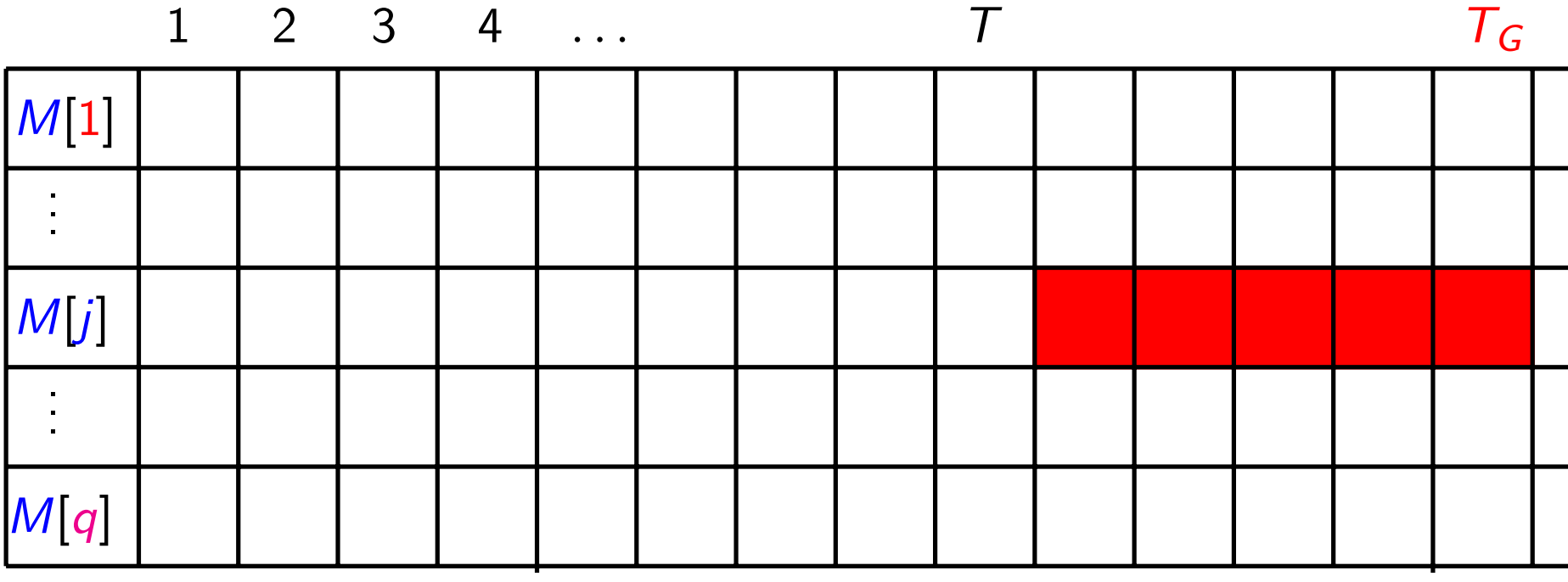
$$\mathrm{opt} \geq \max\{d[1], d[2], \ldots, d[n]\}$$

- Distribuição balanceada:

$$\mathrm{opt} \geq \frac{d[1] + d[2] + \cdots + d[n]}{q}$$

Fator de aproximação

T_G: conclusão do escalonamento obtido pelo algoritmo
i: última tarefa a concluir
j: máquina em que i está, a partir do instante T

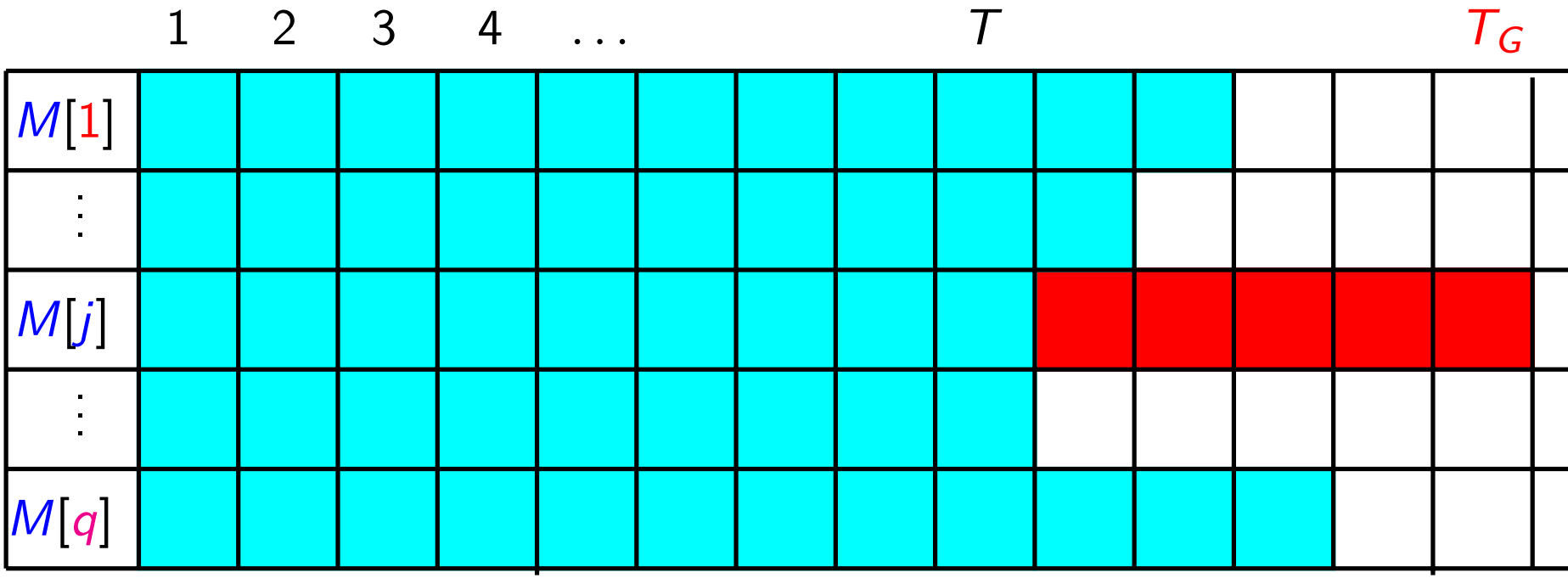


Fator de aproximação (cont.)

T_G: conclusão do escalonamento obtido pelo algoritmo

i: última tarefa a concluir

j: máquina em que i está, a partir do instante T



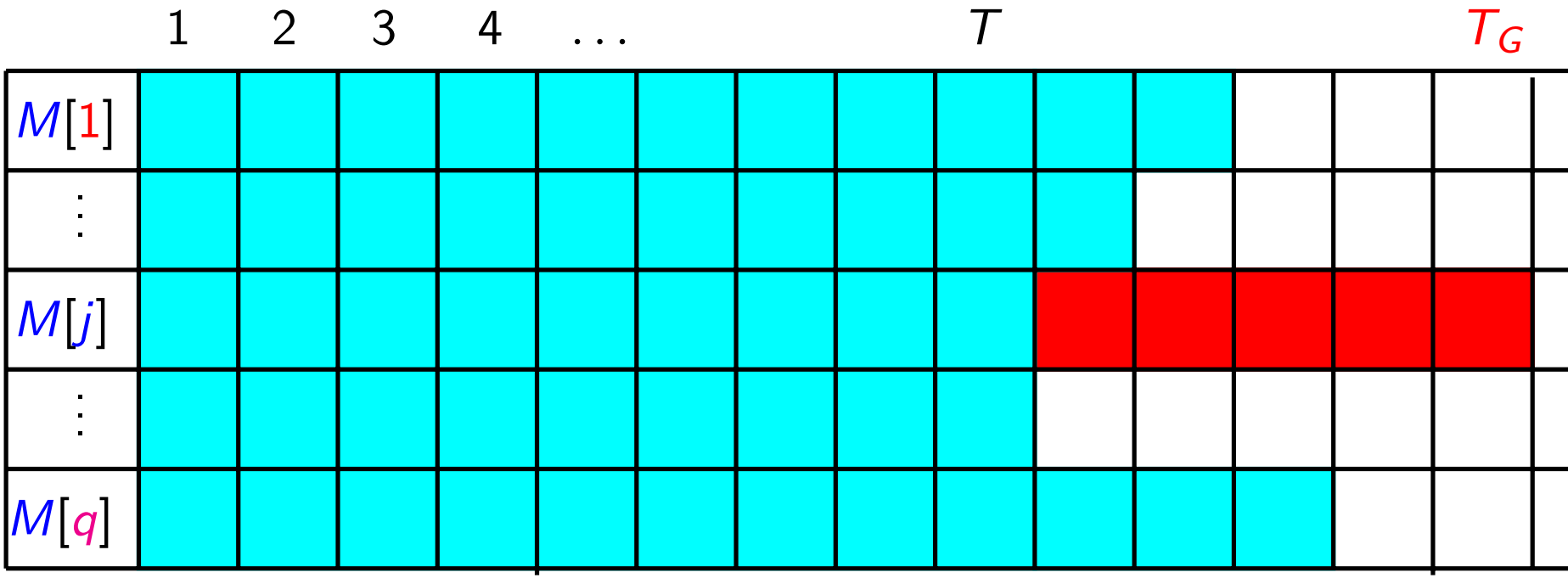
$$T \cdot q \leq d[1] + \cdots + d[n]$$

Fator de aproximação (cont.)

T_G: conclusão do escalonamento obtido pelo algoritmo

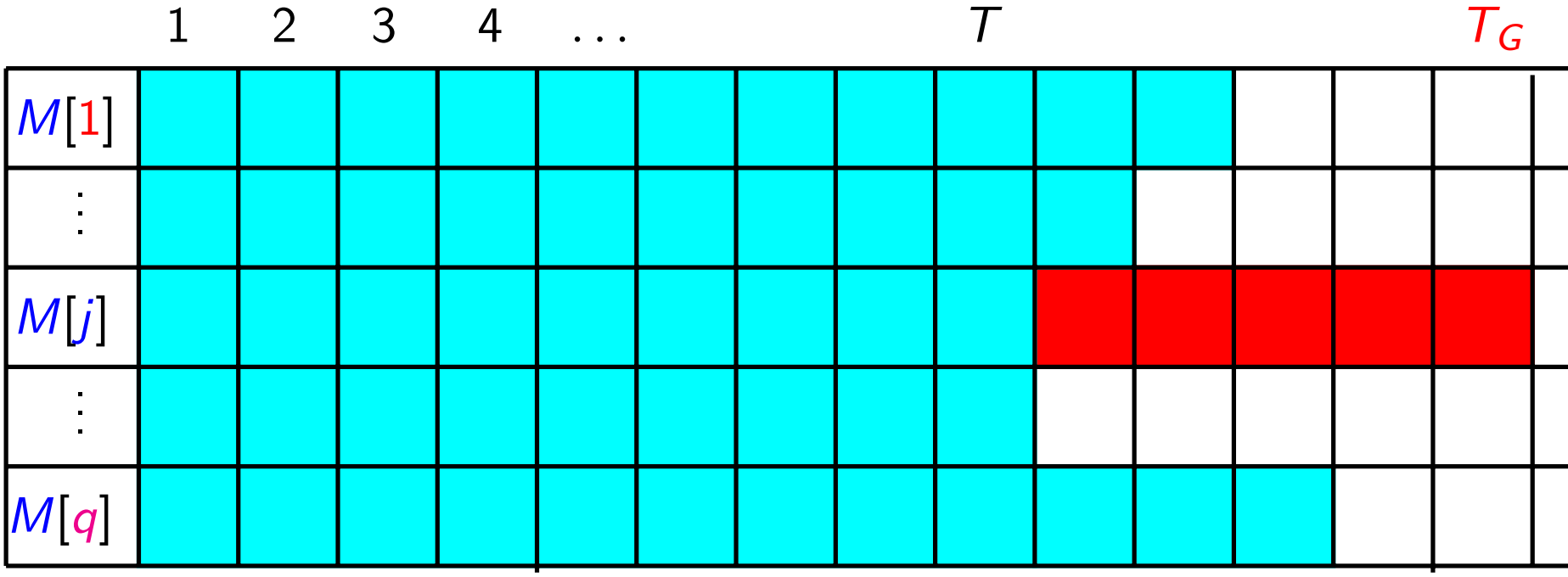
i: última tarefa a concluir

j: máquina em que i está, a partir do instante T



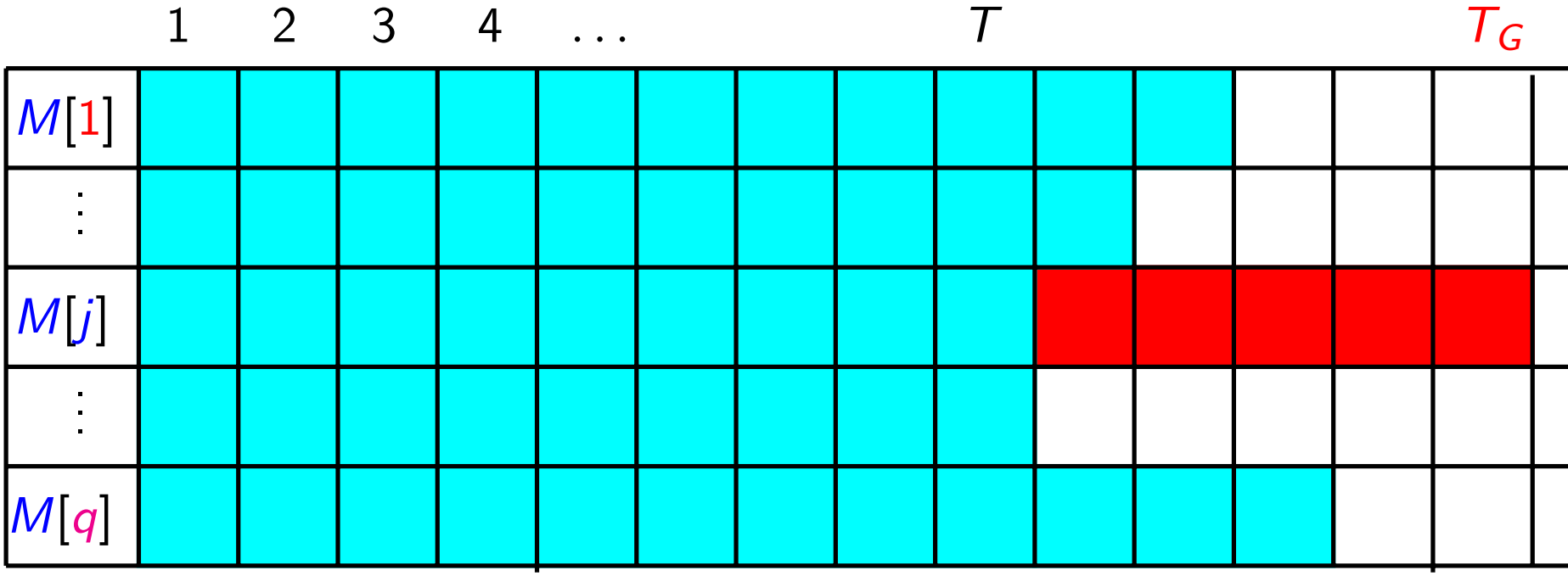
$$T \cdot q \leq d[1] + \cdots + d[n] \quad \Rightarrow \quad T \leq \frac{d[1] + \cdots + d[n]}{q}$$

Fator de aproximação (cont.)



$$T_G = T + d[i]$$

Fator de aproximação (cont.)



$$
\begin{aligned}
T_G &= T + d[i] \\
&\leq \frac{d[1] + \cdots + d[n]}{q} + d[i] \\
&\leq \frac{d[1] + \cdots + d[n]}{q} + \max\{d[1], \ldots, d[n]\} \\
&\leq \mathrm{opt} + \mathrm{opt} = 2\,\mathrm{opt}
\end{aligned}
$$

Conclusão

ESCALONAMENTO-GRAHAM (q, n, d)

```
1   para j ← 1 até q faça
2       M[j] ← ∅
3       T[j] ← 0
4   para i ← 1 até n faça
5       seja k tal que T[k] é mínimo
6       M[k] ← M[k] ∪ {i}
7       T[k] ← T[k] + d[i]
8   devolva {M[1], ..., M[q]}
```

Teorema:

O algoritmo ESCALONAMENTO-GRAHAM é uma 2-aproximação.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Exercício:
Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Exercício:
Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

(Na verdade, ele é uma $\frac{4}{3}$-aproximação.)

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Exercício:
Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

(Na verdade, ele é uma $\frac{4}{3}$-aproximação.)

Existe um PTAS para este problema.

PTAS: polynomial-time approximation scheme

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d_{ij} : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d_{ij} : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Objetivo: encontrar conjunto S com k vértices de V e raio mínimo.

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d_{ij} : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Objetivo: encontrar conjunto S com k vértices de V e raio mínimo.

Vamos mostrar uma 2-aproximação para o problema.

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

O algoritmo acima usaria no máximo k centros.

Esboço da prova:

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

O algoritmo acima usaria no máximo k centros.

Esboço da prova: Cada novo elemento incluído em S leva consigo todo o cluster que o contém no ótimo.

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

O algoritmo acima usaria no máximo k centros.

Esboço da prova: Cada novo elemento incluído em S leva consigo todo o cluster que o contém no ótimo.

Faça uma busca binária pelo valor ótimo.

Mas não precisa do valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$.

GULOSO (n, k, d)

```
1   S ← {1}
2   enquanto |S| < k faça
3       j ← arg max{d(ℓ, S) : ℓ ∈ V}      ▷ mais distante de S
4       S ← S ∪ {j}
5   devolva S
```

Mas não precisa do valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$.

GULOSO (n, k, d)

```
1   S ← {1}
2   enquanto |S| < k faça
3       j ← arg max{d(ℓ, S) : ℓ ∈ V}    ▷ mais distante de S
4       S ← S ∪ {j}
5   devolva S
```

Teorema:
GULOSO é uma 2-aproximação para o problema dos k-centros.

Esboço da prova:

Mas não precisa do valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$.

GULOSO (n, k, d)

```
1   S ← {1}
2   enquanto |S| < k faça
3       j ← arg max{d(ℓ, S) : ℓ ∈ V}      ▷ mais distante de S
4       S ← S ∪ {j}
5   devolva S
```

Teorema:
GULOSO é uma 2-aproximação para o problema dos k-centros.

Esboço da prova: É um caso particular do anterior.
Especificamente, o j escolhido aqui poderia ser
um s do algoritmo anterior em cada iteração.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para
o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que
todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema: Dado um grafo G e um inteiro $k \geq 0$,
existe um conjunto dominante em G de tamanho no máximo k?

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema: Dado um grafo G e um inteiro $k \geq 0$, existe um conjunto dominante em G de tamanho no máximo k?

Este problema é NP-completo.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema: Dado um grafo G e um inteiro $k \geq 0$, existe um conjunto dominante em G de tamanho no máximo k?

Este problema é NP-completo.

Esboço da prova do teorema: Redução do conjunto dominante.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema: Dado um grafo G e um inteiro $k \geq 0$, existe um conjunto dominante em G de tamanho no máximo k?

Este problema é NP-completo.

Esboço da prova do teorema: Redução do conjunto dominante.

Dado G e k, defina $d_{ij} = 1$ se i e j são vizinhos em G, e $d_{ij} = 2$ caso contrário.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P = NP.

Conjunto dominante: conjunto S de vértices tal que todo vértice do grafo ou está em S ou é adjacente a um vértice de S.

Problema: Dado um grafo G e um inteiro $k \geq 0$, existe um conjunto dominante em G de tamanho no máximo k?

Este problema é NP-completo.

Esboço da prova do teorema: Redução do conjunto dominante.

Dado G e k, defina $d_{ij} = 1$ se i e j são vizinhos em G, e $d_{ij} = 2$ caso contrário.

Há conjunto dominante com k vértices sse a α-aproximação aplicada a $(V(G), k, d)$ encontra um tal conjunto dominante.