

Algoritmos Parametrizados

Introdução

Lehilton Pedrosa

Segundo Semestre de 2016

Instituto de Computação – Unicamp

1. Introdução
2. Complexidade
3. Análise multivariada
4. Algoritmo parametrizado
5. Definições

Introdução

Computação é a ciência de resolver problemas!

Computação é a ciência de resolver problemas!

*“Beating **heart** of Computing is **algorithms and complexity**”*

Downey, Fellows

Que são problemas?

Uma maneira (muito baixo nível) para defini-los:

Que são problemas?

Uma maneira (muito baixo nível) para defini-los:

Problema

Alfabeto: conjunto finito Σ de símbolos

Que são problemas?

Uma maneira (muito baixo nível) para defini-los:

Problema

Alfabeto: conjunto finito Σ de símbolos

Relação: propriedade bem definida R entre pares de Σ^*

Que são problemas?

Uma maneira (muito baixo nível) para defini-los:

Problema

Alfabeto: conjunto finito Σ de símbolos

Relação: propriedade bem definida R entre pares de Σ^*

Entrada: instância $x \in \Sigma^*$

Que são problemas?

Uma maneira (muito baixo nível) para defini-los:

Problema

Alfabeto: conjunto finito Σ de símbolos

Relação: propriedade bem definida R entre pares de Σ^*

Entrada: instância $x \in \Sigma^*$

Saída: solução $y \in \Sigma^*$ tal que xRy

Objetivos

- Encontrar uma solução qualquer

Objetivos

- Encontrar uma solução qualquer
- Encontrar uma solução com maior ou menor valor associado

Objetivos

- Encontrar uma solução qualquer
- Encontrar uma solução com maior ou menor valor associado
- Decidir se existe qualquer solução

Objetivos

- Encontrar uma solução qualquer
- Encontrar uma solução com maior ou menor valor associado
- Decidir se existe qualquer solução
- etc.

Objetivos

- Encontrar uma solução qualquer
- Encontrar uma solução com maior ou menor valor associado
- Decidir se existe qualquer solução
- etc.

São problemas de busca, otimização, decisão...

Objetivos

- Encontrar uma solução qualquer
- Encontrar uma solução com maior ou menor valor associado
- **Decidir se existe qualquer solução**
- etc.

São problemas de busca, otimização, **decisão**...

Um exemplo: *race condition*

Descrição:

Um conjunto de processos precisa ser executado, mas alguns deles compartilham os mesmos arquivos. Se dois processos acessarem o mesmo arquivos simultaneamente, pode haver colisão.

Um exemplo: *race condition*

Descrição:

Um conjunto de processos precisa ser executado, mas alguns deles compartilham os mesmos arquivos. Se dois processos acessarem o mesmo arquivos simultaneamente, pode haver colisão.

Podemos fazer diversas perguntas:

Um exemplo: *race condition*

Descrição:

Um conjunto de processos precisa ser executado, mas alguns deles compartilham os mesmos arquivos. Se dois processos acessarem o mesmo arquivos simultaneamente, pode haver colisão.

Podemos fazer diversas perguntas:

1. **Otimização:** Qual o menor número k de processos eu preciso enfileirar?

Um exemplo: *race condition*

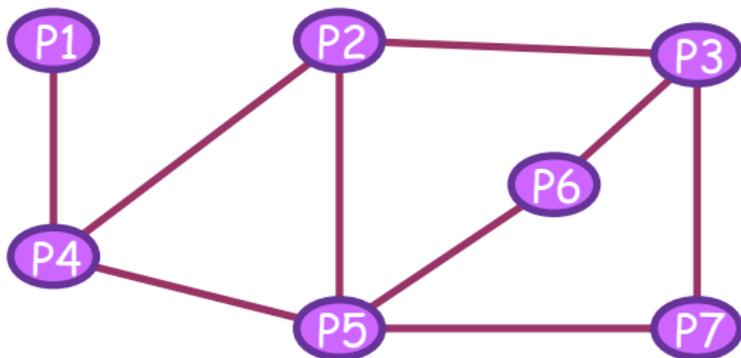
Descrição:

Um conjunto de processos precisa ser executado, mas alguns deles compartilham os mesmos arquivos. Se dois processos acessarem o mesmo arquivos simultaneamente, pode haver colisão.

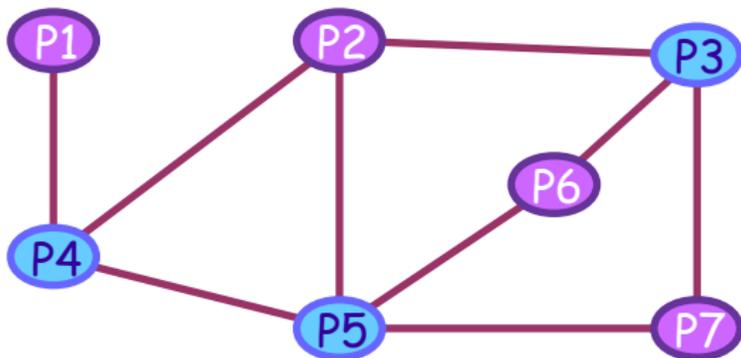
Podemos fazer diversas perguntas:

1. **Otimização:** Qual o menor número k de processos eu preciso enfileirar?
2. **Decisão:** É possível enfileirar só k processos?

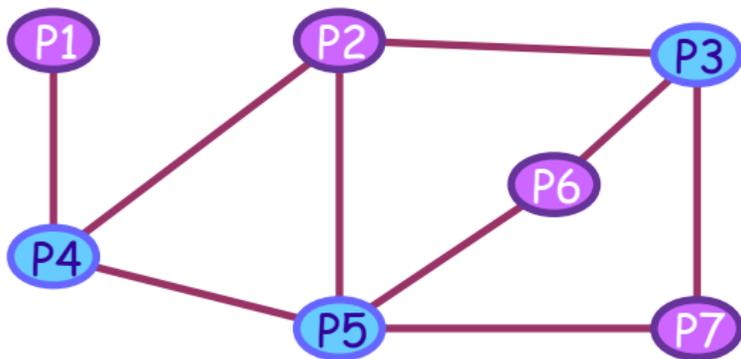
Melhor com um desenho



Melhor com um desenho



Melhor com um desenho



Problema mais conhecido como **Cobertura por vértices**

EXISTECOBERTURA(G, k):

1. Para cada subconjunto $C \subseteq V(G)$:
 - Se $|C| = k$ e $G - C$ não contém arestas,
 - responda **sim**
2. Responda **não**

Algoritmo natural

EXISTECOBERTURA(G, k):

1. Para cada subconjunto $C \subseteq V(G)$:
 - Se $|C| = k$ e $G - C$ não contém arestas,
 - responda **sim**
2. Responda **não**

Complexidade exponencial: $\mathcal{O}(2^n)$, onde $n = |V(G)|$.

EXISTECOBERTURA(G, k):

1. Para cada subconjunto $C \subseteq V(G)$:
 - Se $|C| = k$ e $G - C$ não contém arestas,
 - responda **sim**
2. Responda **não**

Complexidade exponencial: $\mathcal{O}(2^n)$, onde $n = |V(G)|$.

Melhorando um pouco : testamos só $\binom{n}{k}$ possibilidades

Complexidade

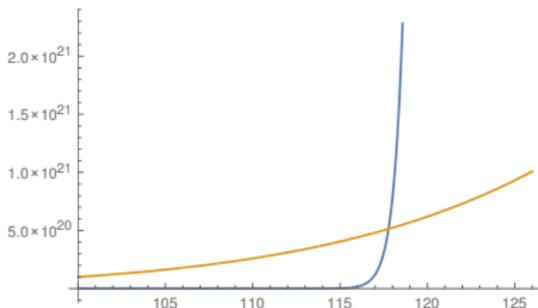
Eficiente = polinomial

Suponha que queremos enfileirar 100 processos, $k = 100$

Eficiente = polinomial

Suponha que queremos enfileirar 100 processos, $k = 100$

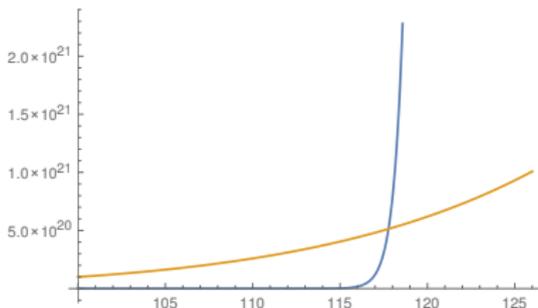
Comparando $\binom{n}{k}$ e um polinômio grande n^{10} :



Eficiente = polinomial

Suponha que queremos enfileirar 100 processos, $k = 100$

Comparando $\binom{n}{k}$ e um polinômio grande n^{10} :



Moral

Lento: $n^n, n!, n^{\frac{n}{100}}, 2^n, n^{\log(n)}$

“Eficiente”: $n, n^2, n^{\mathcal{O}(1)}$

the RAND Combinatorial Symposium during the summer of 1961. I am indebted to many people, at the Symposium and at the National Bureau of Standards, who have taken an interest in the matching problem. There has been much animated discussion on possible versions of an algorithm.

2. Digression. An explanation is due on the use of the words “efficient algorithm.” First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or “code.”

For practical purposes computational details are vital. However, my purpose here is to describe a method that there is an efficient algorithm that is adequate in operation in the sense that it will solve the problem. Perhaps

PATHS, TREES, AND FLOWERS

JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elements called *vertices* and a finite set of elements called *edges* such that each edge *meets* exactly two vertices, called the *end-points* of the edge. An edge is said to *join* its end-points.

Definição (Linguagem)

Uma **linguagem** L é um conjunto de seqüências finitas de um alfabeto sigma: $L \subseteq \Sigma^*$.

Definição (Linguagem)

Uma **linguagem** L é um conjunto de seqüências finitas de um alfabeto sigma: $L \subseteq \Sigma^*$.

Exemplos:

Definição (Linguagem)

Uma **linguagem** L é um conjunto de sequências finitas de um alfabeto sigma: $L \subseteq \Sigma^*$.

Exemplos:

- Pares = $\{x \in \{0,1\}^* : x \text{ termina com } 0\}$

Definição (Linguagem)

Uma **linguagem** L é um conjunto de sequências finitas de um alfabeto sigma: $L \subseteq \Sigma^*$.

Exemplos:

- Pares = $\{x \in \{0, 1\}^* : x \text{ termina com } 0\}$
- VC = conjunto de codificações em binário de $\langle G, k \rangle$, tal que G contém uma cobertura por vértices de tamanho k

Definição (Linguagem)

Uma **linguagem** L é um conjunto de sequências finitas de um alfabeto sigma: $L \subseteq \Sigma^*$.

Exemplos:

- Pares = $\{x \in \{0, 1\}^* : x \text{ termina com } 0\}$
- VC = conjunto de codificações em binário de $\langle G, k \rangle$, tal que G contém uma cobertura por vértices de tamanho k

Linguagem = Problema de decisão

Lembrando: classes de complexidade

Dado problema $Q \subseteq \Sigma^*$ e instância $x \in Q$, o tamanho da instância, n , é o comprimento de x , isso é,

$$n := |x|.$$

Lembrando: classes de complexidade

Dado problema $Q \subseteq \Sigma^*$ e instância $x \in Q$, o tamanho da instância, n , é o comprimento de x , isso é,

$$n := |x|.$$

Definição (Classe \mathcal{P})

$Q \in \mathcal{P}$ se existe algoritmo A , polinomial em n , tal que
 $A(x) = \text{sim}$ sss $x \in Q$.

Lembrando: classes de complexidade

Dado problema $Q \subseteq \Sigma^*$ e instância $x \in Q$, o tamanho da instância, n , é o comprimento de x , isso é,

$$n := |x|.$$

Definição (Classe \mathcal{P})

$Q \in \mathcal{P}$ se existe algoritmo A , polinomial em n , tal que $A(x) = \text{sim}$ sss $x \in Q$.

Definição (Classe \mathcal{NP})

$Q \in \mathcal{NP}$ se existe algoritmo **não-determinístico** A , polinomial em n , tal que:

- $\forall x \in Q, A(x) = \text{sim}$ em alguma execução de A ;
- $\forall x \notin Q, A(x) = \text{não}$.

Definição (Redução)

Dados problemas P e Q , dizemos que P se reduz a Q , $P \leq Q$, se houver uma transformação $T(x)$ que executa em tempo polinomial em $|x|$, tal que, $x \in P$ sss $T(x) \in Q$.

Lembrando: redução

Definição (Redução)

Dados problemas P e Q , dizemos que P se reduz a Q , $P \propto Q$, se houver uma transformação $T(x)$ que executa em tempo polinomial em $|x|$, tal que, $x \in P$ sss $T(x) \in Q$.

Definição (\mathcal{NP} -difícil)

Um problema Q é \mathcal{NP} -difícil se se para todo $P \in \mathcal{NP}$, tivermos $P \propto Q$.

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices,

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices, Cobertura por conjuntos,

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices, Cobertura por conjuntos,
- Caixeiro viajante,

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices, Cobertura por conjuntos,
- Caixeiro viajante, Bipartição,

Problemas \mathcal{NP} -difíceis

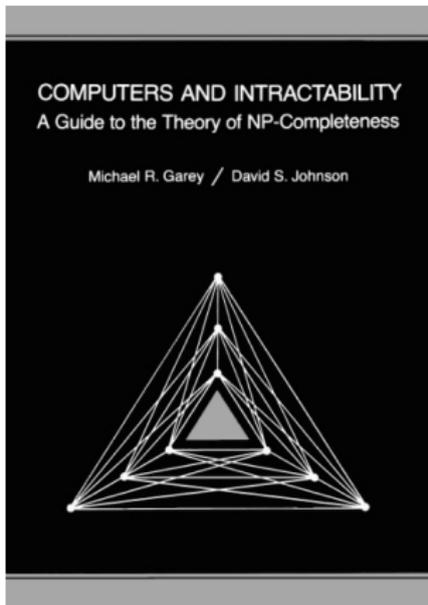
Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices, Cobertura por conjuntos,
- Caixeiro viajante, Bipartição, Soma de subconjunto

Problemas \mathcal{NP} -difíceis

Se é \mathcal{NP} -difícil, então é “pelo menos tão difícil” quanto

- Cobertura por vértices, Cobertura por conjuntos,
- Caixeiro viajante, Bipartição, Soma de subconjunto...



- **Algoritmos de Aproximação:** obtêm uma solução rapidamente, mas comprovadamente não muito ruins
- **Heurísticas e buscas locais:** Tentam obter a melhor solução de um conjunto razoável, mas com tempo limitado
- **Branch and Bound:** Não limita o tempo, mas tenta encontrar a solução exata nas instâncias em que aplicaremos
- **Algoritmos Parametrizados** Sacrifica o tempo de execução, que pode ser exponencial, mas garante que a dependência exponencial está restrita a um parâmetro

Análise multivariada

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10$

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Testando um bilhão de possibilidades por segundo!

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Testando um bilhão de possibilidades por segundo!

n	$\binom{n}{k}$	tempo gasto
-----	----------------	-------------

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Testando um bilhão de possibilidades por segundo!

n	$\binom{n}{k}$	tempo gasto
100	$\binom{100}{10} \approx 1,73 \times 10^{13}$	mais de 4 horas

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Testando um bilhão de possibilidades por segundo!

n	$\binom{n}{k}$	tempo gasto
100	$\binom{100}{10} \approx 1,73 \times 10^{13}$	mais de 4 horas
1000	$\binom{1000}{10} \approx 2,63 \times 10^{23}$	8 milhões de anos

Motivação: nem tão ruim assim

Na prática, o número de processos que precisamos enfileirar é “pequeno”

Pequeno: menor que uma constante **fixa**

Exemplo: $k \leq 10 \Rightarrow$ algoritmo polinomial: $\binom{n}{k} \leq \mathcal{O}(n^k)$

Rápido?

Testando um bilhão de possibilidades por segundo!

n	$\binom{n}{k}$	tempo gasto
100	$\binom{100}{10} \approx 1,73 \times 10^{13}$	mais de 4 horas
1000	$\binom{1000}{10} \approx 2,63 \times 10^{23}$	8 milhões de anos

Separando a dependência de k e n

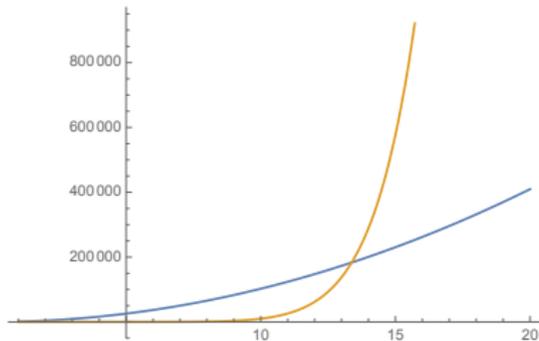
- Suponha $k = 10$

Separando a dependência de k e n

- Suponha $k = 10$
- Comparando $2^k n^2$ versus $\frac{n^k}{10000000}$:

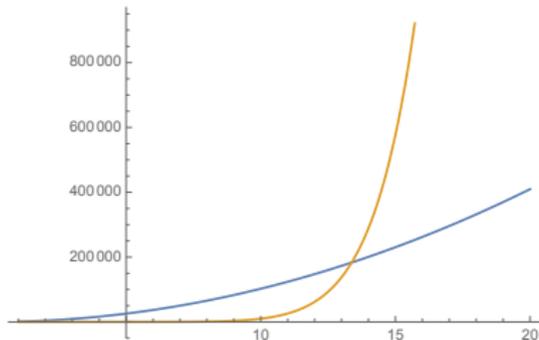
Separando a dependência de k e n

- Suponha $k = 10$
- Comparando $2^k n^2$ versus $\frac{n^k}{10000000}$:



Separando a dependência de k e n

- Suponha $k = 10$
- Comparando $2^k n^2$ versus $\frac{n^k}{10000000}$:



Queremos funções do primeiro tipo!

Problemas parametrizados

Definição (Problema parametrizado)

Um **problema parametrizado** $P \subseteq \Sigma^* \times \mathbb{N}$ é um problema de decisão em que cada instância contém um **parâmetro** k associado, que é um número inteiro.

- Representamos uma instância como $\langle x, k \rangle \in P$
- O parâmetro k é uma medida da instância x

Problemas parametrizados

Definição (Problema parametrizado)

Um **problema parametrizado** $P \subseteq \Sigma^* \times \mathbb{N}$ é um problema de decisão em que cada instância contém um **parâmetro** k associado, que é um número inteiro.

- Representamos uma instância como $\langle x, k \rangle \in P$
- O parâmetro k é uma medida da instância x

Exemplos de parâmetros:

Problemas parametrizados

Definição (Problema parametrizado)

Um **problema parametrizado** $P \subseteq \Sigma^* \times \mathbb{N}$ é um problema de decisão em que cada instância contém um **parâmetro** k associado, que é um número inteiro.

- Representamos uma instância como $\langle x, k \rangle \in P$
- O parâmetro k é uma medida da instância x

Exemplos de parâmetros:

- um **valor explícito** na instância:
 - o tamanho na Cobertura por vértices
 - a tamanho da mochila

Problemas parametrizados

Definição (Problema parametrizado)

Um **problema parametrizado** $P \subseteq \Sigma^* \times \mathbb{N}$ é um problema de decisão em que cada instância contém um **parâmetro** k associado, que é um número inteiro.

- Representamos uma instância como $\langle x, k \rangle \in P$
- O parâmetro k é uma medida da instância x

Exemplos de parâmetros:

- um **valor explícito** na instância:
 - o tamanho na Cobertura por vértices
 - a tamanho da mochila
- uma **medida estrutural** da instância:
 - propriedades de grafo: largura arbórea, grau máximo
 - tamanho mínimo de um item

Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo.

Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

- o expoente **não** depende de k

Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

- o expoente **não** depende de k
- consideramos subproblemas do problema original

Análise multivariada

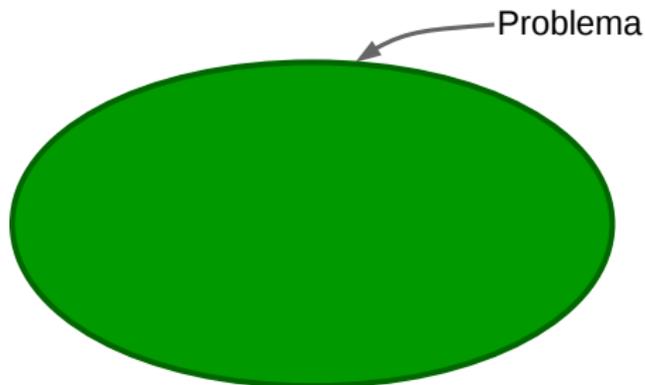
Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

- o expoente **não** depende de k
- consideramos subproblemas do problema original

Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

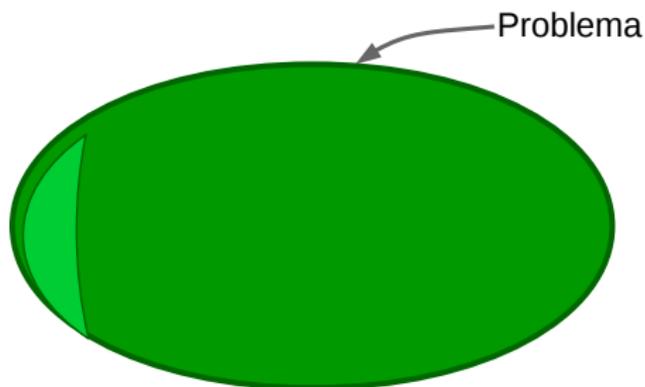
- o expoente **não** depende de k
- consideramos subproblemas do problema original



Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

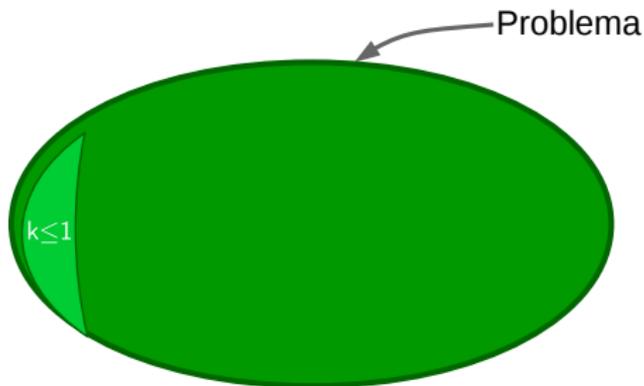
- o expoente **não** depende de k
- consideramos subproblemas do problema original



Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

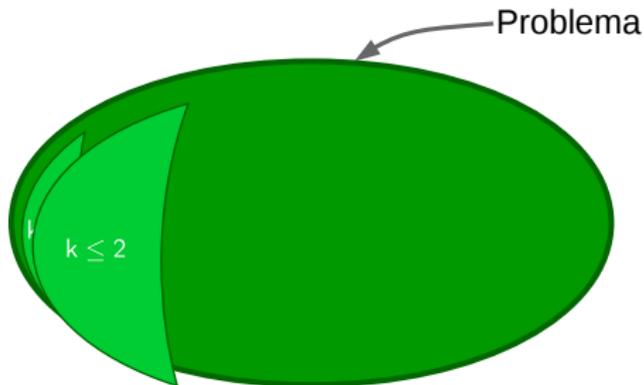
- o expoente **não** depende de k
- consideramos subproblemas do problema original



Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

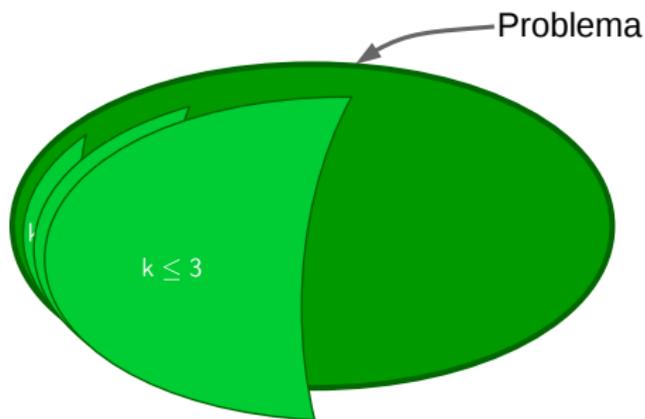
- o expoente **não** depende de k
- consideramos subproblemas do problema original



Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

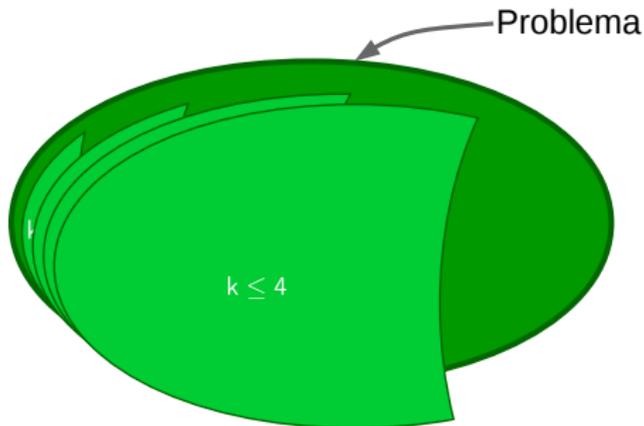
- o expoente **não** depende de k
- consideramos subproblemas do problema original



Análise multivariada

Informalmente: um problema é **tratável por fixação de parâmetro (FPT)** se tiver algoritmo com tempo $O(n^c)$, polinomial para cada k fixo. Ideia:

- o expoente **não** depende de k
- consideramos subproblemas do problema original



Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Entrada: x

```
1110110010101100011111000011
0101101110000010001110010001
0000111010000011101111110100
1000011001111000110010010100
0011000111010101110011011111
0101011000111111011111000011
0101010111011100110000110101
1110000010100110011000101101
1010011100110000011000100111
```

Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Entrada: x , $n := |x|$ bits

```
1110110010101100011111000011
0101101110000010001110010001
0000111010000011101111110100
1000011001111000110010010100
0011000111010101110011011111
0101011000111111011111000011
0101010111011100110000110101
1110000010100110011000101101
1010011100110000011000100111
```

Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Entrada: x , $n := |x|$ bits

```
1110110010101100011111000011
0101101110000010001110010001
0000111010000011101111110100
100001100111100011001010100
00110001110101011001101111
0101011000111111011111000011
0101010111011100110000110101
1110000010100110011000101101
1010011100110000011000100111
```

Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Entrada: x , $n := |x|$ bits

```
1110110010101100011111000011
0101101110000010001110010001
00001110100000111011110100
100001100111100011001010100
001100011101010111001101111
0101011000111111011111000011
0101010111011100110000110101
1110000010100110011000101101
1010011100110000011000100111
```

Isolando a dificuldade

Ideia principal: isolamos o parâmetro que contribui fundamentalmente para a dificuldade:

Entrada: x , $n := |x|$ bits

```
1110110010101100011111000011
0101101110000010001110010001
00001110100000111011110100
100001100111100011001010100
00110001110101011001101111
0101011000111111011111000011
010101011101110011000110101
1110000010100110011000101101
1010011100110000011000100111
```

Parte difícil: parâmetro k

Uma mudança de paradigma

- 1992: trabalhos de (in)tratabilidade com parâmetros fixos

Uma mudança de paradigma

- 1992: trabalhos de (in)tratabilidade com parâmetros fixos
- 1999: livro de Downey e Fellows

Uma mudança de paradigma

- 1992: trabalhos de (in)tratabilidade com parâmetros fixos
- 1999: livro de Downey e Fellows

Muitos recursos disponíveis

- Diversos livros
- Wiki de parametrizados: <http://fpt.wikidot.com/>
- Escolas: <http://fptschool.mimuw.edu.pl/>

“The future of algorithmics is multivariate”

Downey, Fellows

O futuro?

“The future of algorithmics is multivariate”

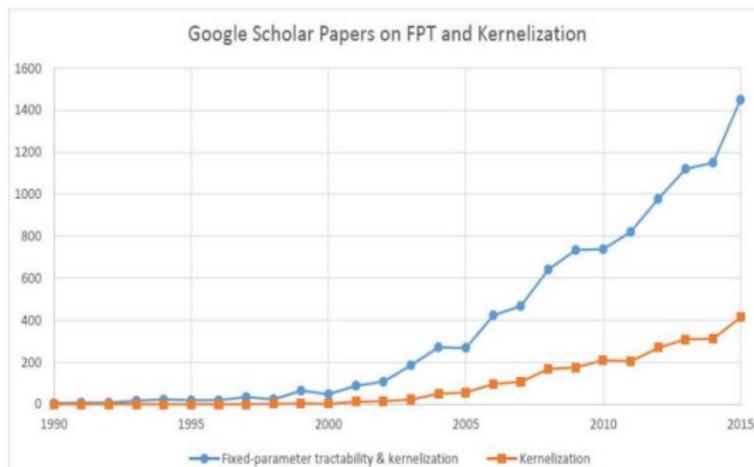
Downey, Fellows

- Conferências especializadas: IPEC
- Diversos artigos: ICALP, STOC, FOCS, ESA, LATIN...

“The future of algorithmics is multivariate”

Downey, Fellows

- Conferências especializadas: IPEC
- Diversos artigos: ICALP, STOC, FOCS, ESA, LATIN...



(Fonte: Bart Jansen, <http://fpt.wikidot.com/>)

Algoritmo parametrizado

Voltando ao nosso problema

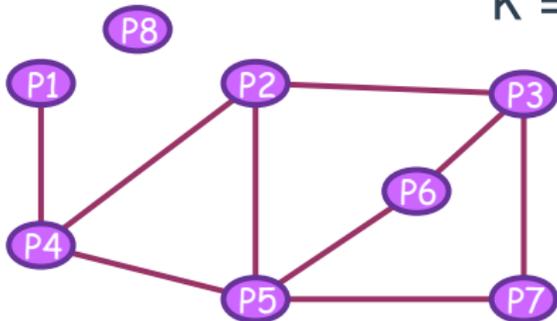
Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$k = 3$

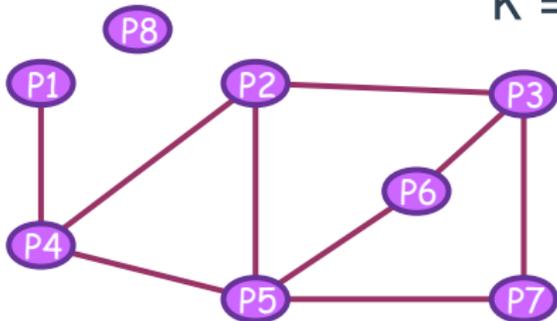


Perguntas:

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$k = 3$

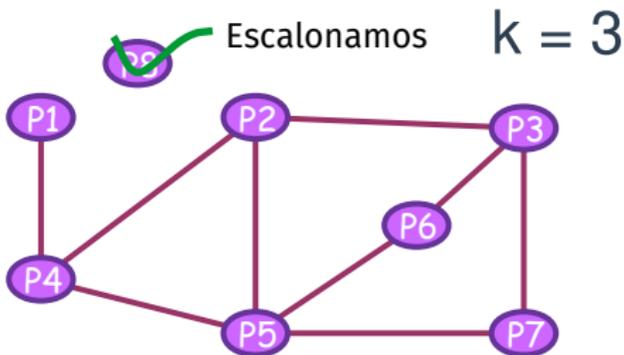


Perguntas:

1. O processo P8 será enfileirado?

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos



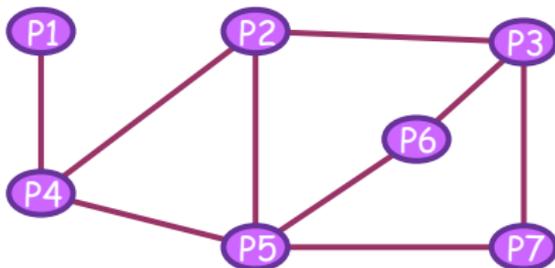
Perguntas:

1. O processo P8 será enfileirado?
 - **não**, um vértice isolado não tem conflito k^2 arestas

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$$k = 3$$



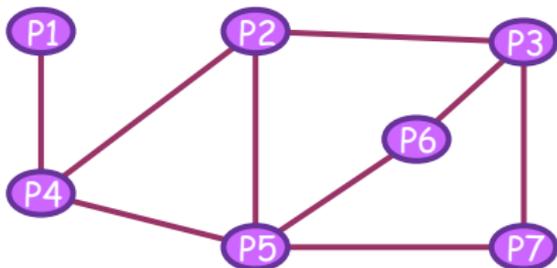
Perguntas:

1. O processo P8 será enfileirado?
 - não, um vértice isolado não tem conflito k^2 arestas

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$$k = 3$$



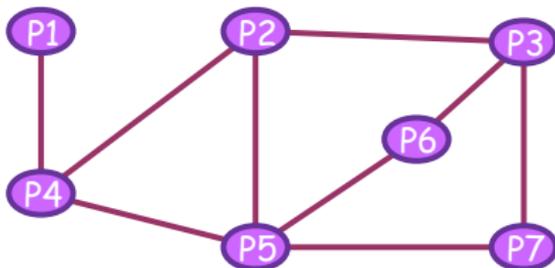
Perguntas:

1. O processo P8 será enfileirado?
 - não, um vértice isolado não tem conflito k^2 arestas
2. Existe solução com valor $k = 3$

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$$k = 3$$



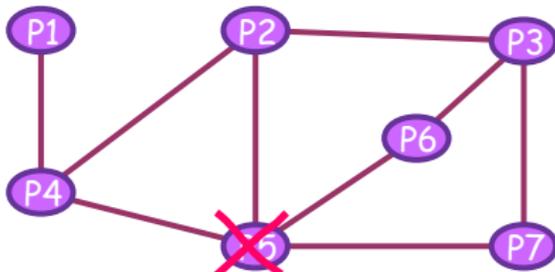
Perguntas:

1. O processo P8 será enfileirado?
 - não, um vértice isolado não tem conflito k^2 arestas
2. Existe solução com valor $k = 3$, mas que não enfileira P5?

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

Enfileiramos e diminuimos k $k = 3$



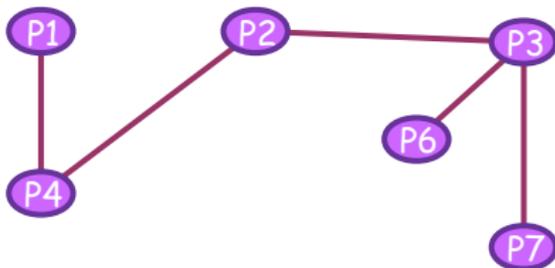
Perguntas:

1. O processo P8 será enfileirado?
 - não, um vértice isolado não tem conflito k^2 arestas
2. Existe solução com valor $k = 3$, mas que não enfileira P5?
 - não, P5 tem 4 conflitos!

Voltando ao nosso problema

Melhor algoritmo até agora: 8 milhões de anos

$k = 2$



Perguntas:

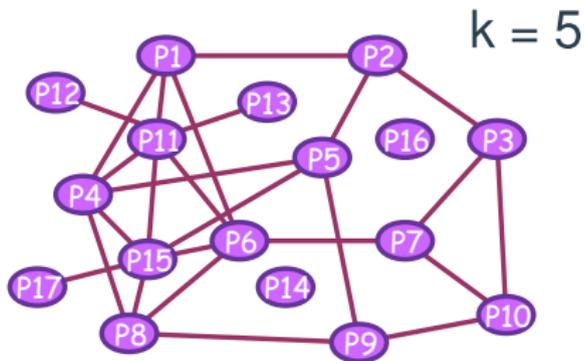
1. O processo P8 será enfileirado?
 - não, um vértice isolado não tem conflito k^2 arestas
2. Existe solução com valor $k = 3$, mas que não enfileira P5?
 - não, P5 tem 4 conflitos!

Voltando ao nosso problema

Agora podemos simplificar uma instância dada.

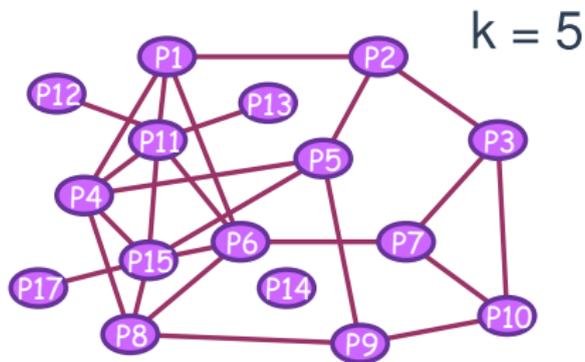
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



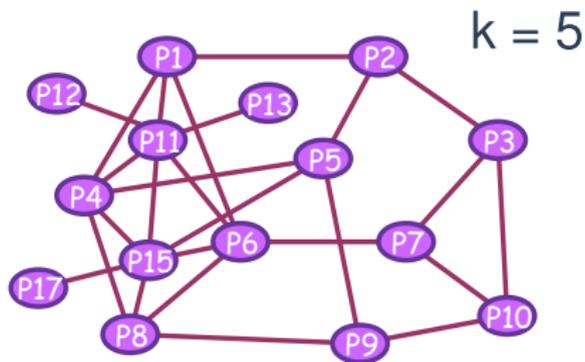
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



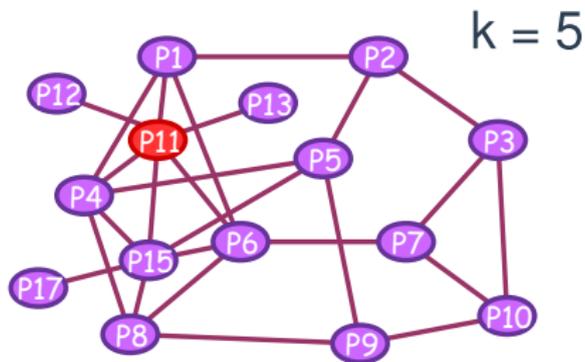
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



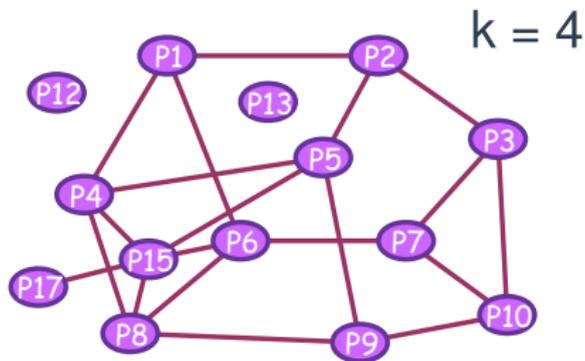
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



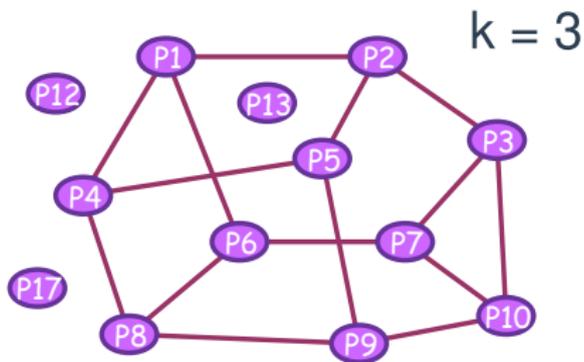
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



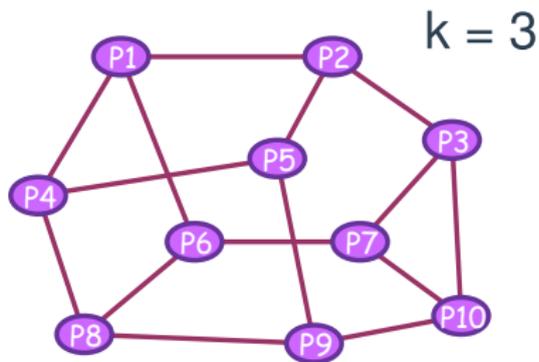
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



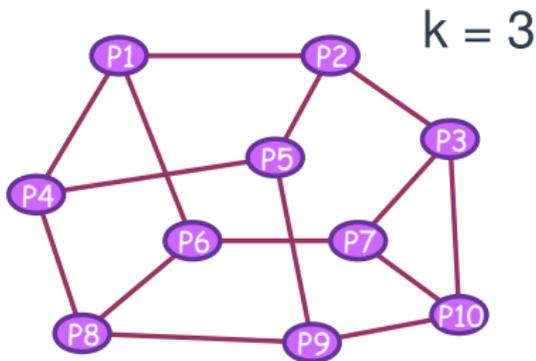
Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



Voltando ao nosso problema

Agora podemos simplificar uma instância dada.



Mais uma pergunta:

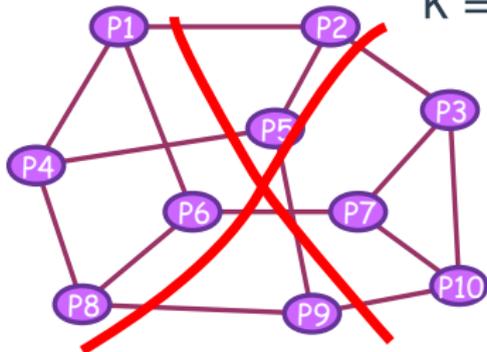
4. Existe solução para essa instância com valor $k = 3$?

Voltando ao nosso problema

Agora podemos simplificar uma instância dada.

Respondemos **não!**

$k = 3$



Mais uma pergunta:

4. Existe solução para essa instância com valor $k = 3$?
 - **não**, uma instância **sim** tem no máximo k^2 arestas

Isolando a dificuldade

Resolvemos a parte “fácil” da instância:

Resolvemos a parte “fácil” da instância:

Resolvemos a parte “fácil” da instância:

1. Removemos vértices isolados

Resolvemos a parte “fácil” da instância:

1. Removemos vértices isolados

- selecione $v \in V(G)$ com $d(v) = 0$
- obtenha nova instância $\langle G - v, k \rangle$

Resolvemos a parte “fácil” da instância:

1. Removemos vértices isolados

- selecione $v \in V(G)$ com $d(v) = 0$
- obtenha nova instância $\langle G - v, k \rangle$

2. Removemos vértices pesados:

Resolvemos a parte “fácil” da instância:

1. Removemos vértices isolados

- selecione $v \in V(G)$ com $d(v) = 0$
- obtenha nova instância $\langle G - v, k \rangle$

2. Removemos vértices pesados:

- selecione $v \in V(G)$ com $d(v) > k$
- obtenha nova instância $\langle G - v, k - 1 \rangle$

Resolvemos a parte “fácil” da instância:

1. Removemos vértices isolados

- selecione $v \in V(G)$ com $d(v) = 0$
- obtenha nova instância $\langle G - v, k \rangle$

2. Removemos vértices pesados:

- selecione $v \in V(G)$ com $d(v) > k$
- obtenha nova instância $\langle G - v, k - 1 \rangle$

3. Decidimos as instâncias densas

O que sobra?

Agora temos um grafo G tal que

O que sobra?

Agora temos um grafo G tal que

- para todo $v \in V(G)$, temos $1 \leq d(v) \leq k$
- número de arestas $|E(G)| \leq k^2$

O que sobra?

Agora temos um grafo G tal que

- para todo $v \in V(G)$, temos $1 \leq d(v) \leq k$
- número de arestas $|E(G)| \leq k^2$

\Rightarrow Vamos usar o **mesmo** algoritmo anterior:

O que sobra?

Agora temos um grafo G tal que

- para todo $v \in V(G)$, temos $1 \leq d(v) \leq k$
- número de arestas $|E(G)| \leq k^2$

⇒ Vamos usar o **mesmo** algoritmo anterior:

- complexidade é $\mathcal{O}\left(\binom{n}{k}\right)$

O que sobra?

Agora temos um grafo G tal que

- para todo $v \in V(G)$, temos $1 \leq d(v) \leq k$
- número de arestas $|E(G)| \leq k^2$

⇒ Vamos usar o **mesmo** algoritmo anterior:

- complexidade é $\mathcal{O}\left(\binom{n}{k}\right)$
- mas agora temos $n \leq 2k^2$

O que sobra?

Agora temos um grafo G tal que

- para todo $v \in V(G)$, temos $1 \leq d(v) \leq k$
- número de arestas $|E(G)| \leq k^2$

⇒ Vamos usar o **mesmo** algoritmo anterior:

- complexidade é $\mathcal{O}\left(\binom{n}{k}\right)$
- mas agora temos $n \leq 2k^2$ (por quê?)

Situação atual

Melhor algoritmo até agora: $\mathcal{O}\left(\binom{2k^2}{k}\right)$

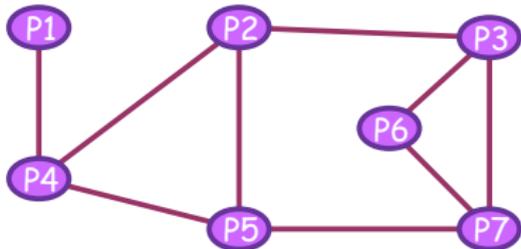
- para $k = 10$, $\approx 2,25 \times 10^{16}$: 8,5 meses

Situação atual

Melhor algoritmo até agora: $\mathcal{O}\left(\binom{2k^2}{k}\right)$

- para $k = 10$, $\approx 2,25 \times 10^{16}$: 8,5 meses

$k = 3$

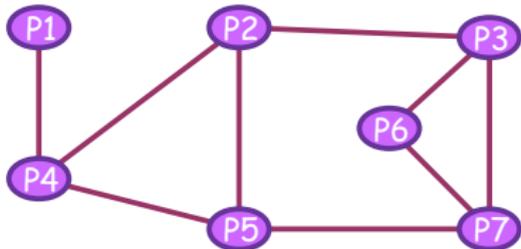


Situação atual

Melhor algoritmo até agora: $\mathcal{O}\left(\binom{2k^2}{k}\right)$

- para $k = 10$, $\approx 2,25 \times 10^{16}$: 8,5 meses

$k = 3$



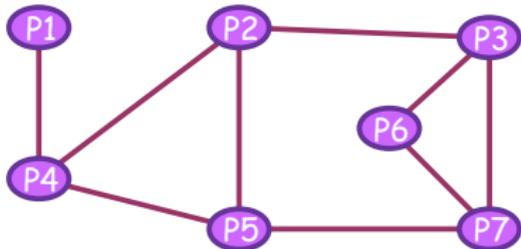
Pergunta: existe solução que não enfileira P1?

Situação atual

Melhor algoritmo até agora: $\mathcal{O}\left(\binom{2k^2}{k}\right)$

- para $k = 10$, $\approx 2,25 \times 10^{16}$: 8,5 meses

$k = 3$



Pergunta: existe solução que não enfileira P1?

- **sim**, sempre existe uma solução que não enfileira v , quando $d(v) = 1$

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- Suficiente?

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Repensando nosso problema:

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Repensando nosso problema:

Como resolver conflitos?

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Repensando nosso problema:

Como resolver conflitos?

- se X e Y conflitam: (1) enfileiramos X ; ou (2) enfileiramos Y

Última melhoria

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Repensando nosso problema:

Como resolver conflitos?

- se X e Y conflitam: (1) enfileiramos X ; ou (2) enfileiramos Y
- podemos enfileirar no máximo k tarefas

Sobram só k^2 vértices: tempo $\mathcal{O}\left(\binom{k^2}{k}\right)$

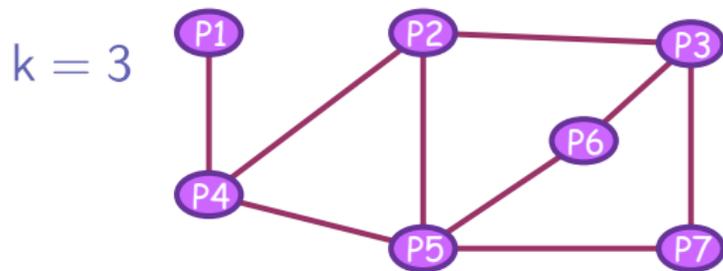
- para $k = 10$, $\approx 1,73 \times 10^{10}$: 4,8 horas
- **Suficiente?** 4,8h não é viável para um escalonador de processo

Repensando nosso problema:

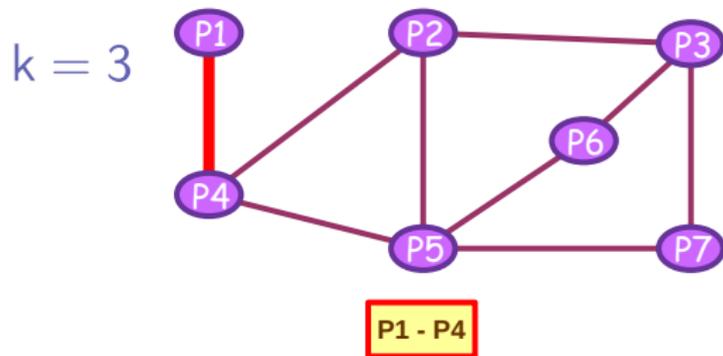
Como resolver conflitos?

- se X e Y conflitam: (1) enfileiramos X ; ou (2) enfileiramos Y
- podemos enfileirar no máximo k tarefas
- **Conclusão:** tomamos no máximo k decisões

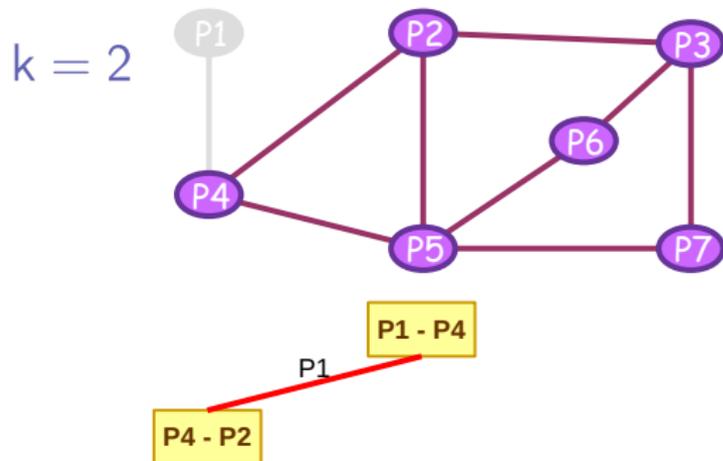
Árvore de decisão



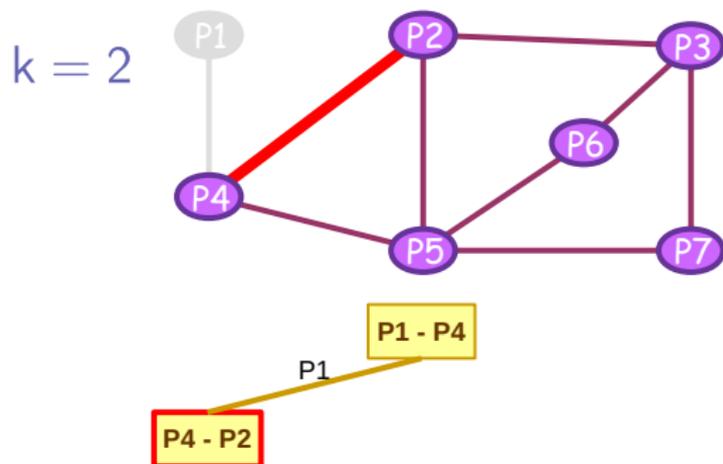
Árvore de decisão



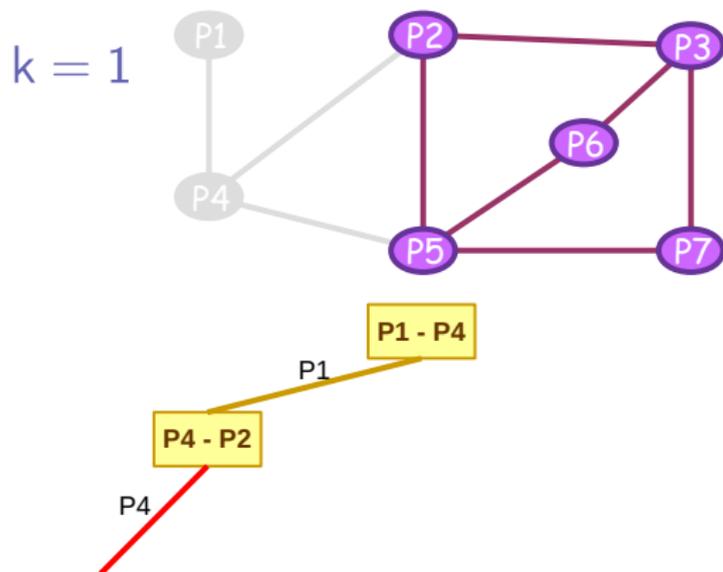
Árvore de decisão



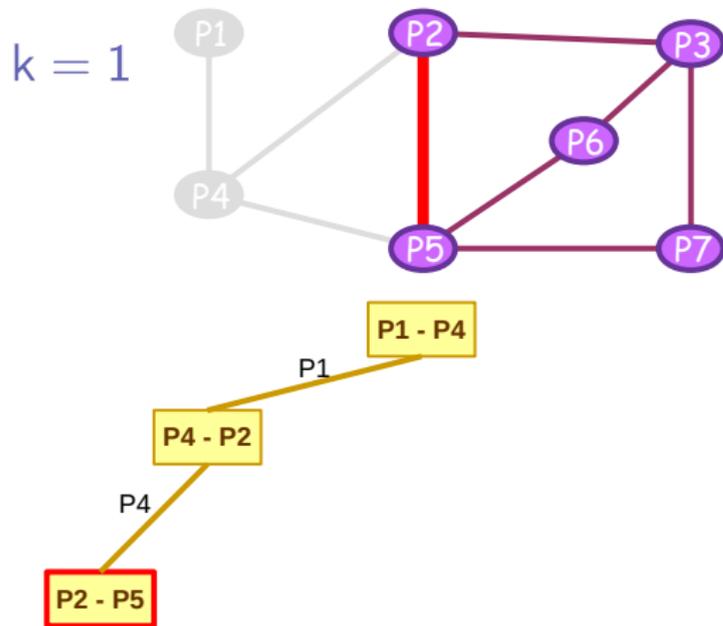
Árvore de decisão



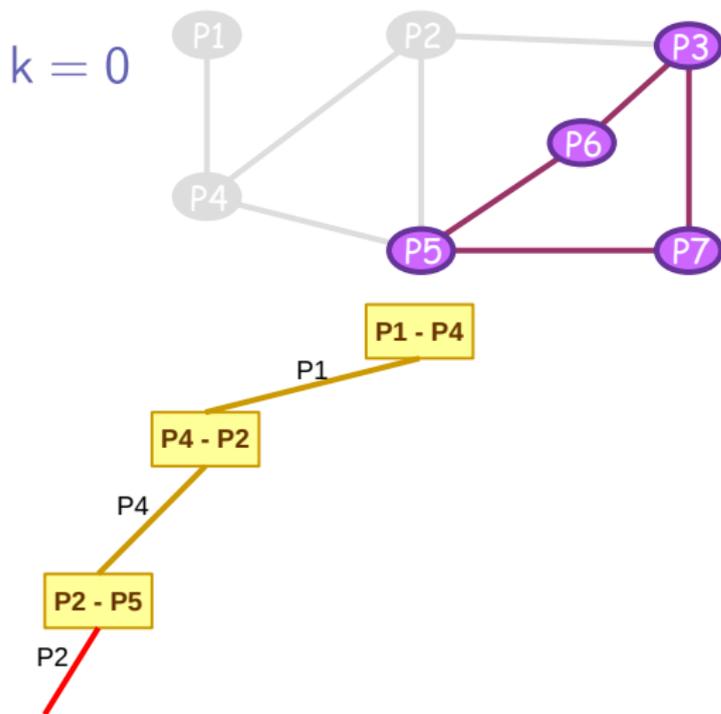
Árvore de decisão



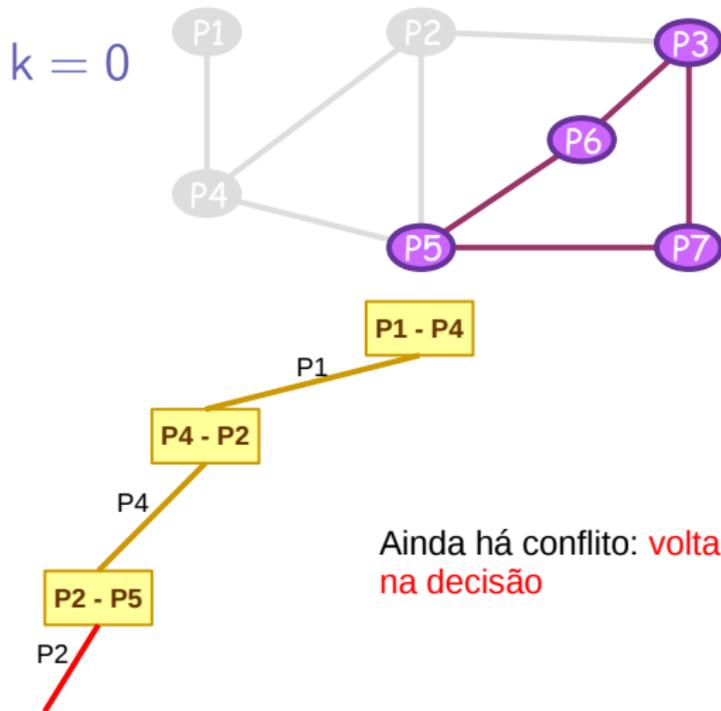
Árvore de decisão



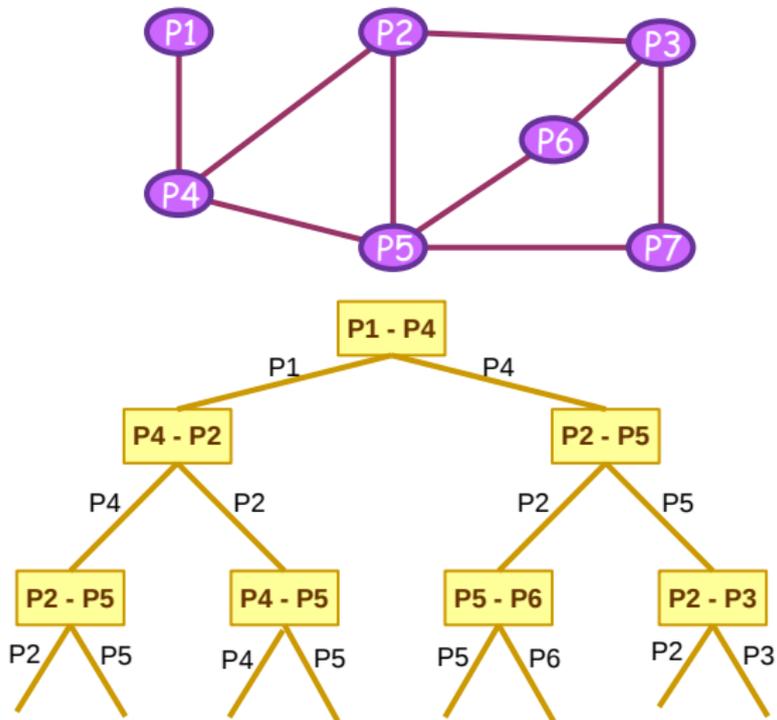
Árvore de decisão



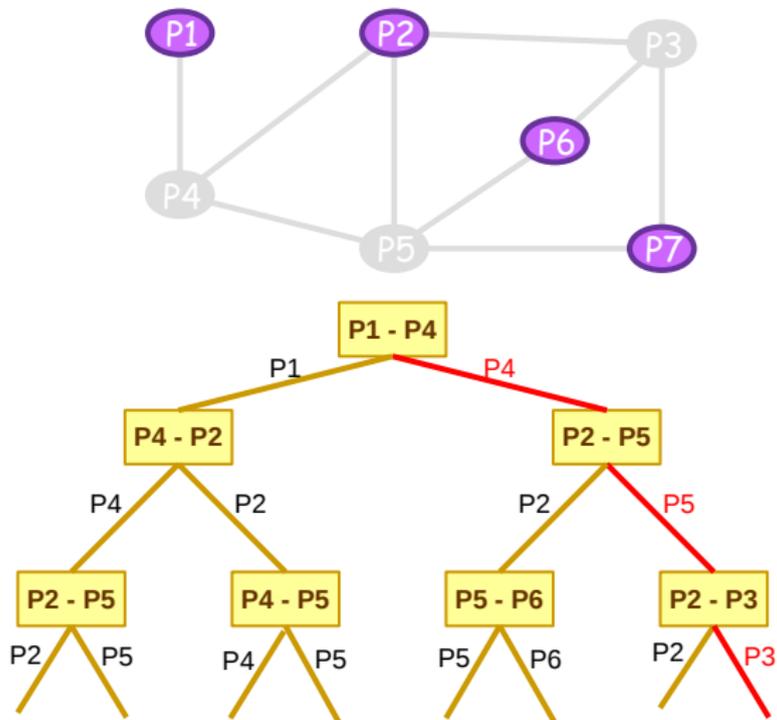
Árvore de decisão



Árvore de decisão



Árvore de decisão



Análise

Análise

- árvore tem altura k

Análise

- árvore tem altura k
- número de nós percorridos é no máximo 2^k

Análise

- árvore tem altura k
- número de nós percorridos é no máximo 2^k
- cada visita pode ser implementada em $\mathcal{O}(|V| + |E|) = \mathcal{O}(nk)$ (para instância reduzida)

Análise

- árvore tem altura k
- número de nós percorridos é no máximo 2^k
- cada visita pode ser implementada em $\mathcal{O}(|V| + |E|) = \mathcal{O}(nk)$ (para instância reduzida)

Resumindo:

- Algoritmo inicial: tempo $\mathcal{O}\left(\binom{n}{k}\right) = \mathcal{O}(n^k)$
- Algoritmo atual: tempo $\mathcal{O}(2^k kn) = \mathcal{O}(n)$

Análise

- árvore tem altura k
- número de nós percorridos é no máximo 2^k
- cada visita pode ser implementada em $\mathcal{O}(|V| + |E|) = \mathcal{O}(nk)$ (para instância reduzida)

Resumindo:

- **Algoritmo inicial:** tempo $\mathcal{O}\left(\binom{n}{k}\right) = \mathcal{O}(n^k)$
- **Algoritmo atual:** tempo $\mathcal{O}(2^k kn) = \mathcal{O}(n)$
 - para $n = 1000$, $k = 10$: uma fração de segundo

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Objetivo:

- Atribuir cada tarefa a um processador

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Objetivo:

- Atribuir cada tarefa a um processador
- equivalentemente, queremos **colorir** os vértices

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Objetivo:

- Atribuir cada tarefa a um processador
 - equivalentemente, queremos **colorir** os vértices
-
- **Fato:** Coloração é \mathcal{NP} -difícil

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Objetivo:

- Atribuir cada tarefa a um processador
 - equivalentemente, queremos **colorir** os vértices
-
- **Fato:** Coloração é \mathcal{NP} -difícil
 - **Alternativa natural:** parametrizar pelo número de cores k

Situação

E se tivermos vários processadores com cópias dos recursos compartilhados pelas tarefas?

Objetivo:

- Atribuir cada tarefa a um processador
 - equivalentemente, queremos **colorir** os vértices
-
- **Fato:** Coloração é \mathcal{NP} -difícil
 - **Alternativa natural:** parametrizar pelo número de cores k
→ não temos tantos processadores assim!

- Sabemos: 3-Coloração também é \mathcal{NP} -difícil

- **Sabemos:** 3-Coloração também é \mathcal{NP} -difícil
⇒ não existe algoritmo $\mathcal{O}(n^c)$ (a menos que $\mathcal{P} = \mathcal{NP}$)

- **Sabemos:** 3-Coloração também é \mathcal{NP} -difícil
 - ⇒ não existe algoritmo $\mathcal{O}(n^c)$ (a menos que $\mathcal{P} = \mathcal{NP}$)
 - ⇒ **portanto:** 3-Coloração não é **FPT** com o número de cores

- **Sabemos:** 3-Coloração também é \mathcal{NP} -difícil
 - ⇒ não existe algoritmo $\mathcal{O}(n^c)$ (a menos que $\mathcal{P} = \mathcal{NP}$)
 - ⇒ **portanto:** 3-Coloração não é **FPT** com o número de cores

Conclusão: se um problema parametrizado com parâmetro fixo k é \mathcal{NP} -difícil, então não esperamos que seja **FPT**.

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par.

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par. **Objetivo:**

- decidir se existe subconjunto $S \subseteq V(G)$, com $|S| = k$, tal que $G[S]$ é completo

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par. **Objetivo:**

- decidir se existe subconjunto $S \subseteq V(G)$, com $|S| = k$, tal que $G[S]$ é completo

Pergunta: Sempre que um problema parametrizado com parâmetro fixo é polinomial, então ele é **FPT**?

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par. **Objetivo:**

- decidir se existe subconjunto $S \subseteq V(G)$, com $|S| = k$, tal que $G[S]$ é completo

Pergunta: Sempre que um problema parametrizado com parâmetro fixo é polinomial, então ele é **FPT**?

- Provavelmente não.

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par. **Objetivo:**

- decidir se existe subconjunto $S \subseteq V(G)$, com $|S| = k$, tal que $G[S]$ é completo

Pergunta: Sempre que um problema parametrizado com parâmetro fixo é polinomial, então ele é **FPT**?

- Provavelmente não.
- **Exemplo:**
 - clique com parâmetro k fixo é polinomial (por quê?)

Outro exemplo: clique

Situação

Queremos identificar um conjunto de tarefas que devem ser serializadas: tem conflito par-a-par. **Objetivo:**

- decidir se existe subconjunto $S \subseteq V(G)$, com $|S| = k$, tal que $G[S]$ é completo

Pergunta: Sempre que um problema parametrizado com parâmetro fixo é polinomial, então ele é **FPT**?

- Provavelmente não.
- **Exemplo:**
 - clique com parâmetro k fixo é polinomial (por quê?)
 - não esperamos um algoritmo **FPT** para clique

Definições

Classe $W[1]$

- problemas que podem ser reduzidos (em tempo **FPT**) para um circuito de *trama 1*

Classe $W[1]$

- problemas que podem ser reduzidos (em tempo **FPT**) para um circuito de *trama 1*
- uma classe muito grande de problemas

Classe $W[1]$

- problemas que podem ser reduzidos (em tempo **FPT**) para um circuito de *trama 1*
- uma classe muito grande de problemas
- formalizaremos no final do curso

Classe $W[1]$

- problemas que podem ser reduzidos (em tempo **FPT**) para um circuito de *trama 1*
- uma classe muito grande de problemas
- formalizaremos no final do curso

Fato: Clique é $W[1]$ -difícil

Classe $W[1]$

- problemas que podem ser reduzidos (em tempo **FPT**) para um circuito de *trama 1*
- uma classe muito grande de problemas
- formalizaremos no final do curso

Fato: Clique é $W[1]$ -difícil

⇒ Não acreditamos que um problema $W[1]$ -difícil seja **FPT**!

Clique

- Instância de tamanho n : \mathcal{NP} -difícil
- Parâmetro k : $\mathbf{W}[1]$ -difícil

Clique

- Instância de tamanho n : \mathcal{NP} -difícil
- Parâmetro k : $\mathbf{W}[1]$ -difícil

Pergunta: Então não existe **FPT** para clique?

Clique

- Instância de tamanho n : \mathcal{NP} -difícil
- Parâmetro k : $\mathbf{W}[1]$ -difícil

Pergunta: Então não existe **FPT** para clique?

Alternativa: ainda podemos explorar mais as instâncias

- uma tarefa não tem muitos conflitos (na prática)!
- o maior grau de um vértice Δ é limitado

Alternativa: análise multivariada

Parametrizamos clique pelo valor de Δ !

Alternativa: análise multivariada

Parametrizamos clique pelo valor de Δ !

Como?

Alternativa: análise multivariada

Parametrizamos clique pelo valor de Δ !

Como?

CLIQUE(G, k):

1. Para cada $v \in V(G)$:
 - Para cada $S \subseteq N[v]$:
 - Se $G[S]$ é uma clique com k vértices:
responda **sim**
2. Responda **não**

Definição (*Fixed Parameter Tractable*)

Dizemos que um problema parametrizado $P \subseteq \Sigma^* \times \mathbb{N}$ é tratável por parâmetro fixo (**FPT**) se, e somente se, existe um algoritmo A , uma constante c e uma função computável f tais que, para todo par $\langle x, k \rangle \in \Sigma^* \times \mathbb{N}$,

1. $A(\langle x, k \rangle)$ executa em tempo $f(k)|x|^c$;
2. $\langle x, k \rangle \in P$ sss $A(\langle x, k \rangle) = 1$.

Definição (*Fixed Parameter Tractable*)

Dizemos que um problema parametrizado $P \subseteq \Sigma^* \times \mathbb{N}$ é tratável por parâmetro fixo (**FPT**) se, e somente se, existe um algoritmo A , uma constante c e uma função computável f tais que, para todo par $\langle x, k \rangle \in \Sigma^* \times \mathbb{N}$,

1. $A(\langle x, k \rangle)$ executa em tempo $f(k)|x|^c$;
2. $\langle x, k \rangle \in P$ sss $A(\langle x, k \rangle) = 1$.

Notação: Para funções $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ e $f : \mathbb{N} \rightarrow \mathbb{N}$, escrevemos:

$g(k, n) = \mathcal{O}^*(f(k))$ sss existe constante c tal que $g(k, n) = \mathcal{O}(f(k) n^c)$.

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k
2. Tentar desenvolver algoritmo para k pequeno

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k
2. Tentar desenvolver algoritmo para k pequeno
 - Se problema é **W[1]**-difícil:

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k
2. Tentar desenvolver algoritmo para k pequeno
 - Se problema é **W[1]**-difícil:
 - procurar outro parâmetro

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k
2. Tentar desenvolver algoritmo para k pequeno
 - Se problema é **W[1]**-difícil:
 - procurar outro parâmetro
 - Se problema é **FPT**:

Perspectiva **FPT** para tratar um problema \mathcal{NP} -difícil

Estratégia

1. Procurar um parâmetro k
2. Tentar desenvolver algoritmo para k pequeno
 - Se problema é **W[1]**-difícil:
 - procurar outro parâmetro
 - Se problema é **FPT**:
 - enriquecer modelo e torná-lo mais realista