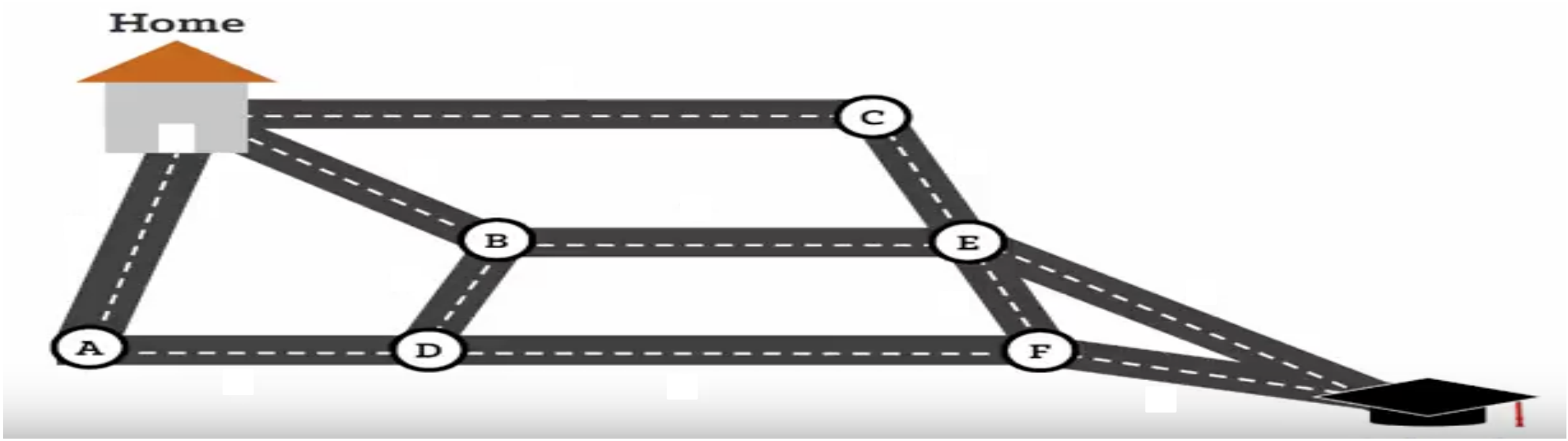
Approximation Schemes for the Restricted Shortest Path Problem

Hauptvortrag. Mai 28, 2018

Mazen Ebada

Betreuer : Tobias Zündorf

INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMICS GROUP



Gliederung

- Definition
- Zwei exakte Pseudopolyomial-Algorithmen
- ”TEST(V)” Hilfsmethode für das erste FPTAS
- Erstes FPTAS (Rounding) von Komplexität :

  $\mathcal{O}(\log(\log(B)) \cdot (|E|(n/\varepsilon) + \log(\log(B))$
- ”Partition” Hilfsmethode für das zweite FPTAS
- Zweites FPTAS (Interval Partitioning) von Komplexität :

  $\mathcal{O}(|E|(n^2/\varepsilon) \cdot \log(n/\varepsilon))$

Institute of Theoretical Informatics
Algorithmics Group

Definition

Gegeben : Graph G = (V,E) mit Knotenmenge V = $\{1, \ldots, n\}$ und Kantenmange E, wobei jede Kante hat länge c_{ij} und Übergangszeit t_{ij} .

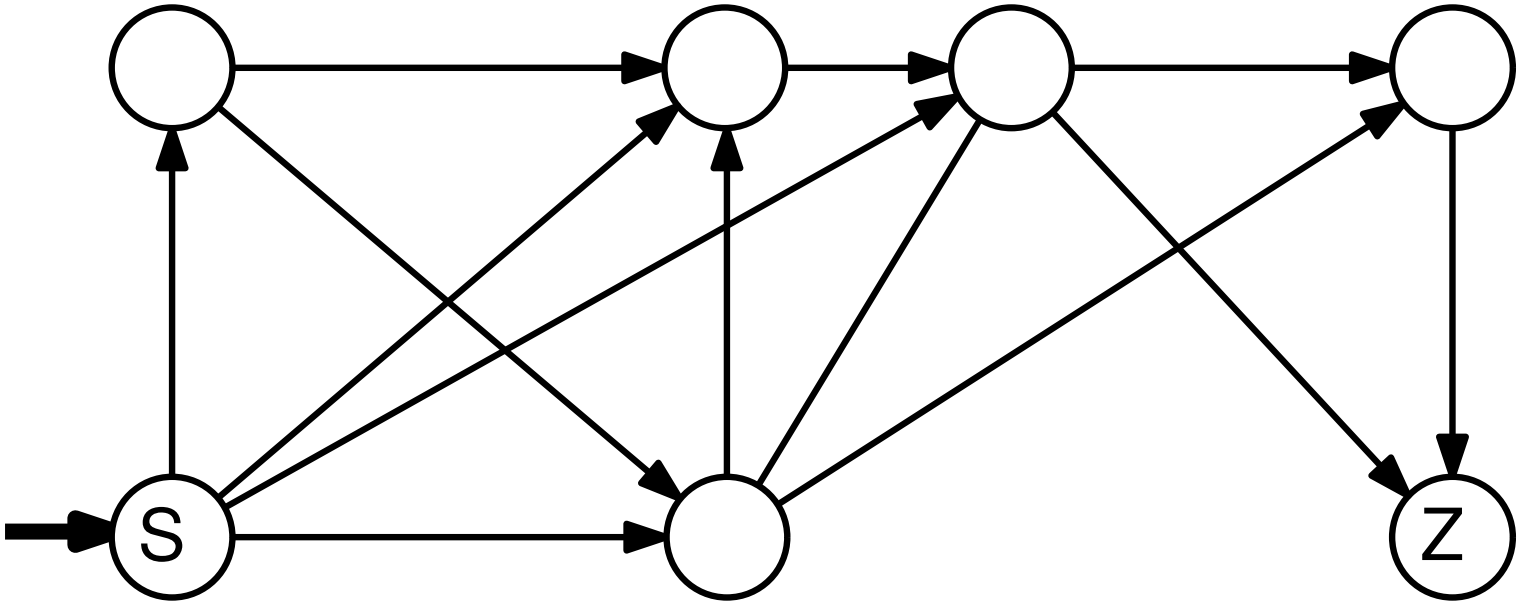
Gesucht : Der kürzeste Weg von einem Knoten zum anderen mit einer Übergangszeit $\leq$ T für ein gegebenes T.

Institute of Theoretical Informatics
Algorithmics Group

Definition

Gegeben : Graph G = (V,E) mit Knotenmenge V = {1, ... ,n} und Kantenmange E, wobei jede Kante hat länge c_{ij} und Übergangszeit t_{ij} .

Gesucht : Der kürzeste Weg von einem Knoten zum anderen mit einer Übergangszeit $\leq$ T für ein gegebenes T.

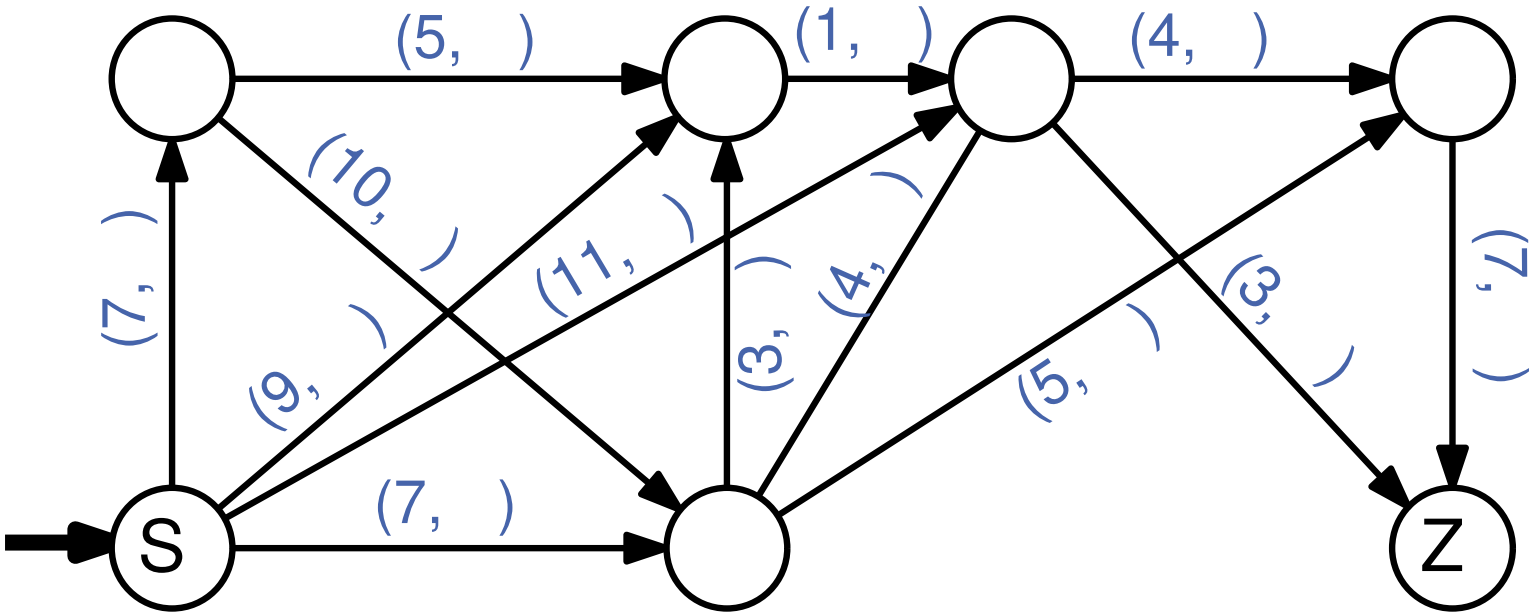


Institute of Theoretical Informatics
Algorithmics Group

Definition

Gegeben : Graph G = (V,E) mit Knotenmenge V = {1, ... ,n} und Kantenmange E, wobei jede Kante hat länge c_{ij} und Übergangszeit t_{ij} .

Gesucht : Der kürzeste Weg von einem Knoten zum anderen mit einer Übergangszeit $\leq$ T für ein gegebenes T.

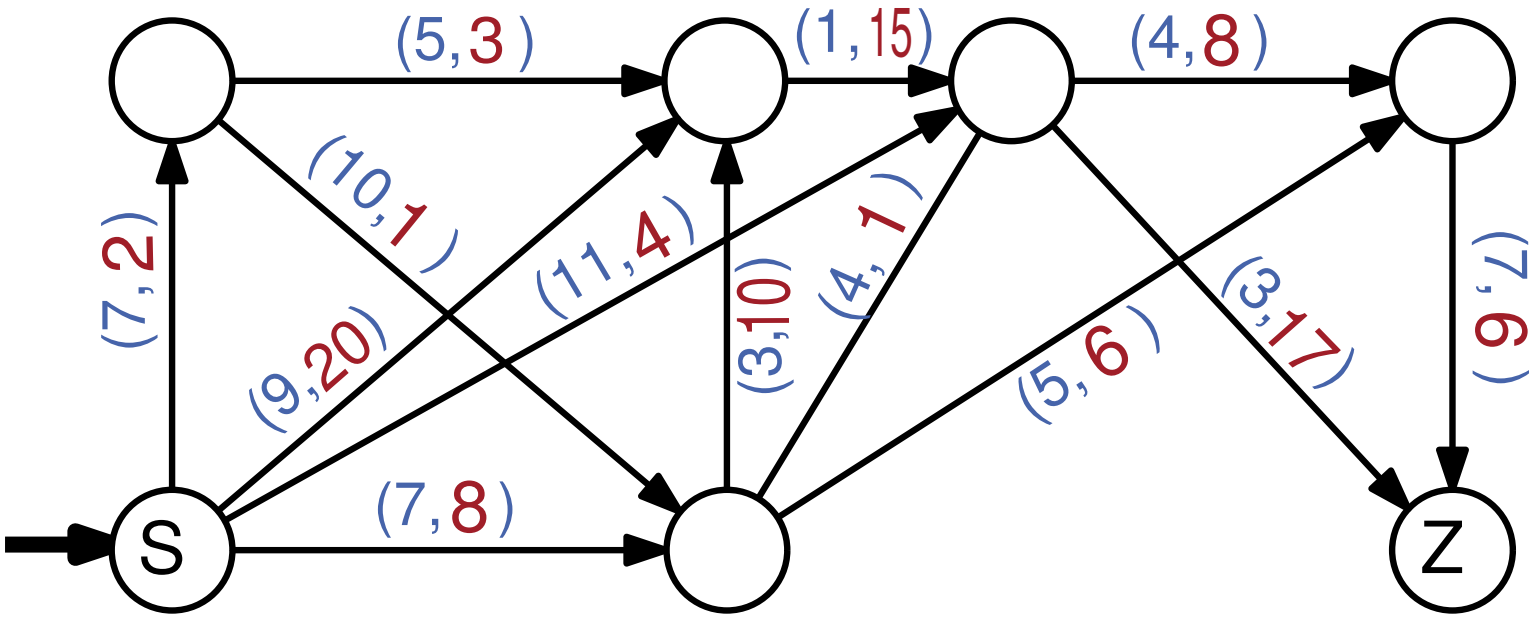


Institute of Theoretical Informatics
Algorithmics Group

Definition

Gegeben : Graph G = (V,E) mit Knotenmenge V = {1, ... ,n} und Kantenmange E, wobei jede Kante hat länge c_{ij} und Übergangszeit t_{ij} .

Gesucht : Der kürzeste Weg von einem Knoten zum anderen mit einer Übergangszeit $\leq$ T für ein gegebenes T.

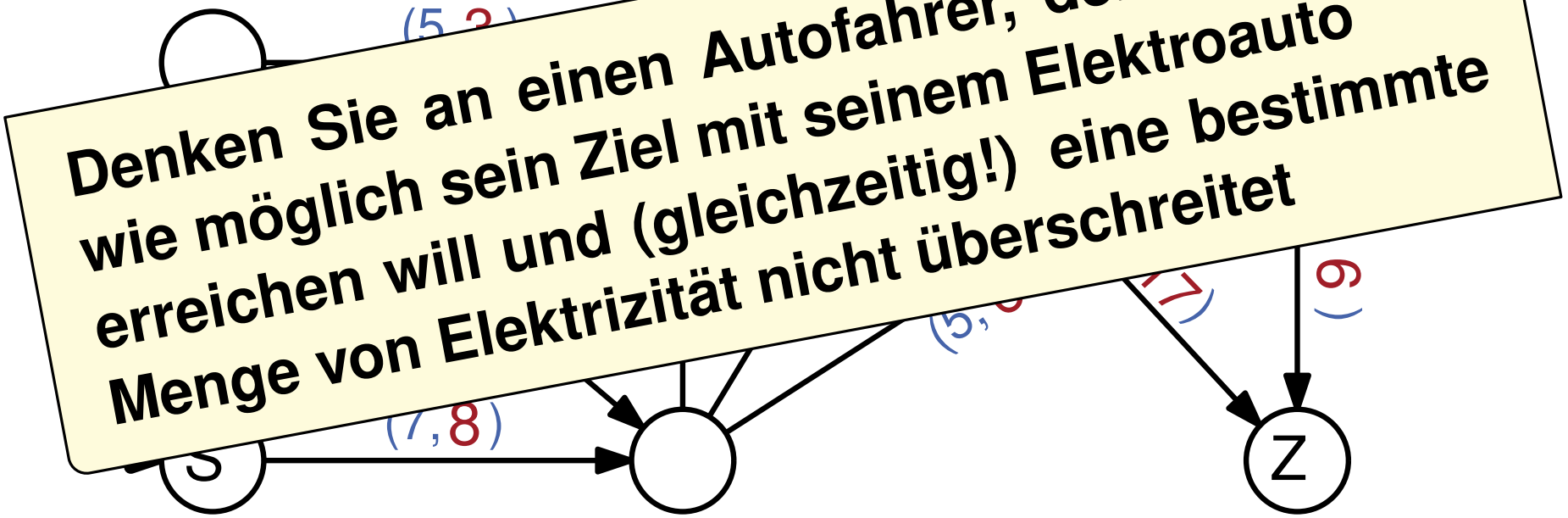


Definition

Gegeben : Graph G = (V,E) mit Knotenmenge V = {1, ... ,n} und Kantenmange E, wobei jede Kante hat länge c_{ij} und Übergangszeit t_{ij} .

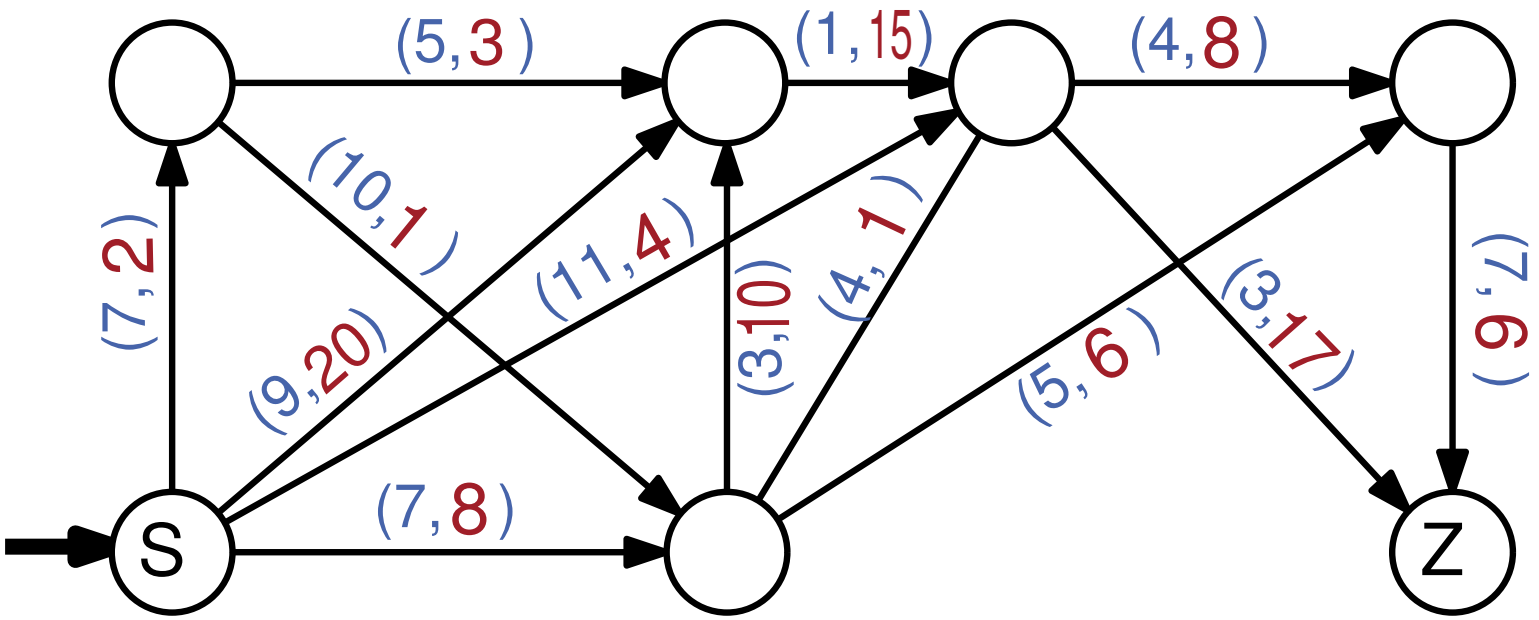
Gesucht : Der kürzeste Weg von einem Knoten zum anderen mit einer Übergangszeit $\leq$ T für ein gegebenes T.

Denken Sie an einen Autofahrer, der so schnell wie möglich sein Ziel mit seinem Elektroauto erreichen will und (gleichzeitig!) eine bestimmte Menge von Elektrizität nicht überschreitet



Institute of Theoretical Informatics
Algorithmics Group

Definition - (2)



Shortest Path Problem

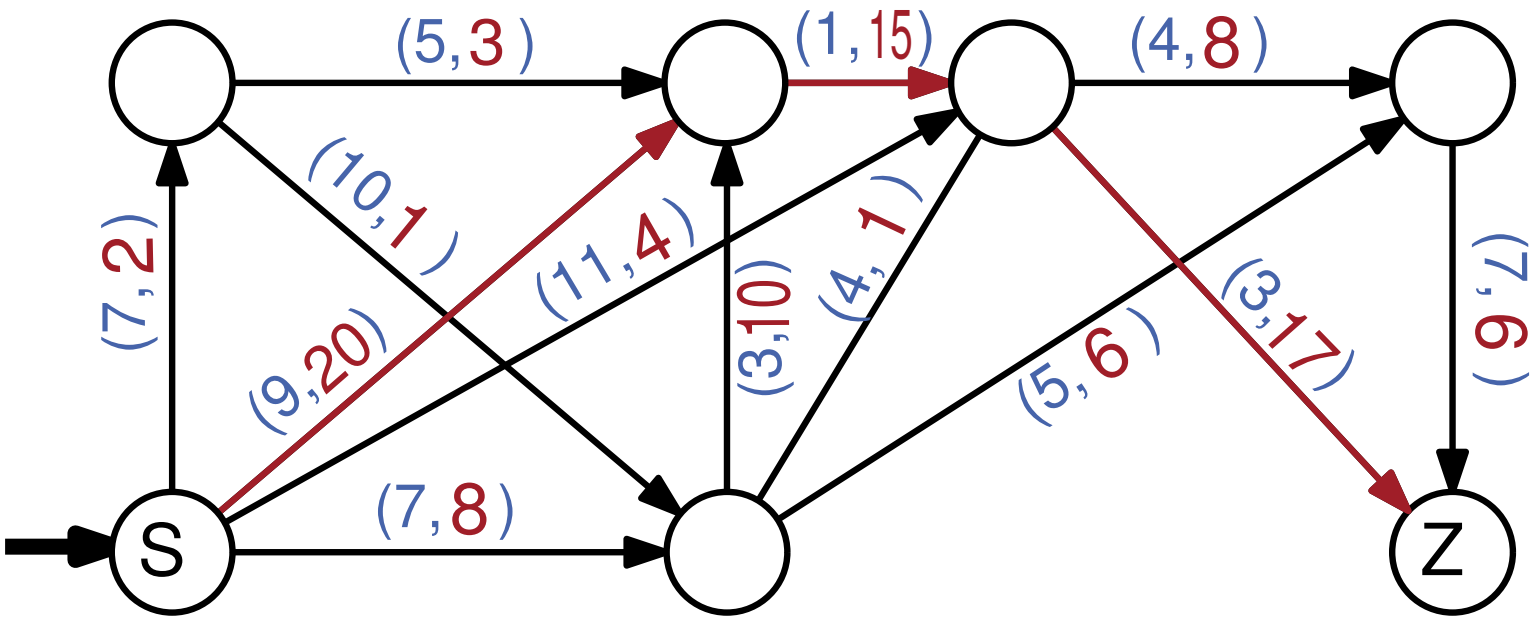
Restricted Shortest Path Problem With T = 25

Restricted Shortest Path Problem With T = 20

...

Institute of Theoretical Informatics
Algorithmics Group

Definition - (2)



Shortest Path Problem

OPT = 13

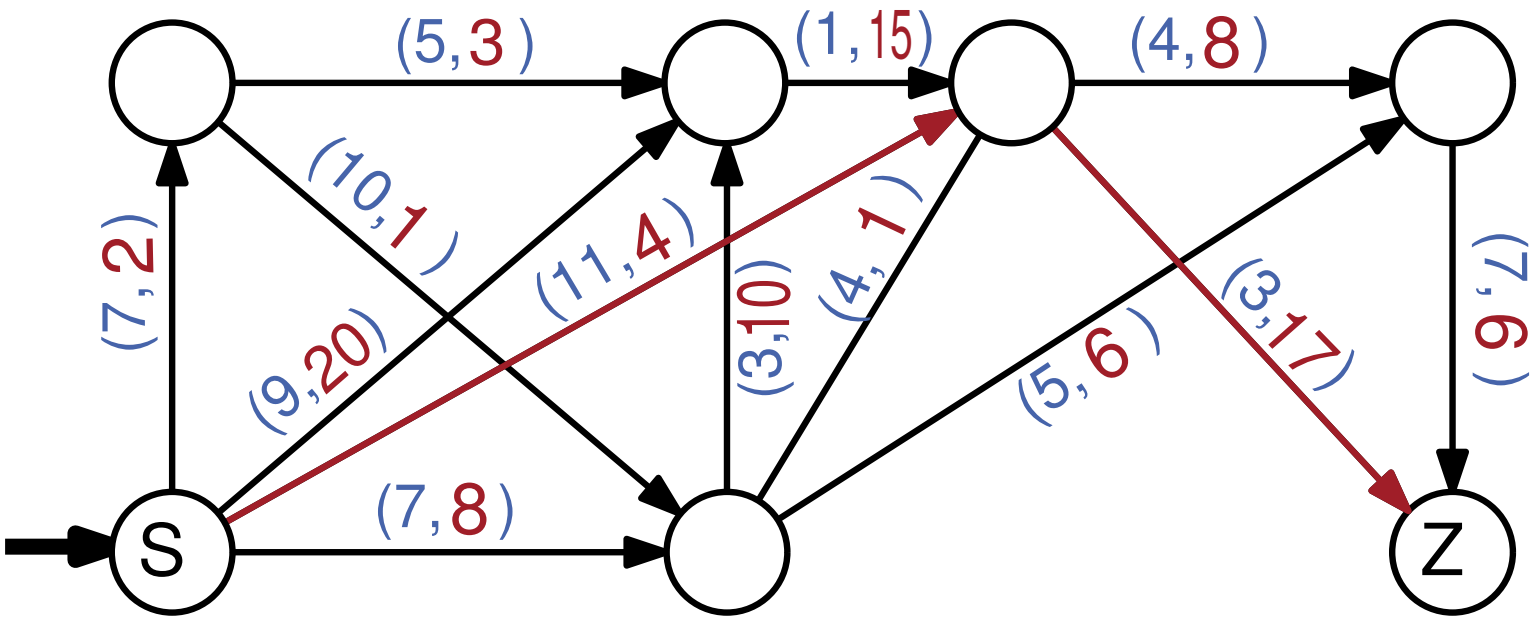
Restricted Shortest Path Problem With T = 25

Restricted Shortest Path Problem With T = 20

...

Institute of Theoretical Informatics
Algorithmics Group

Definition - (2)



Shortest Path Problem — OPT = 13

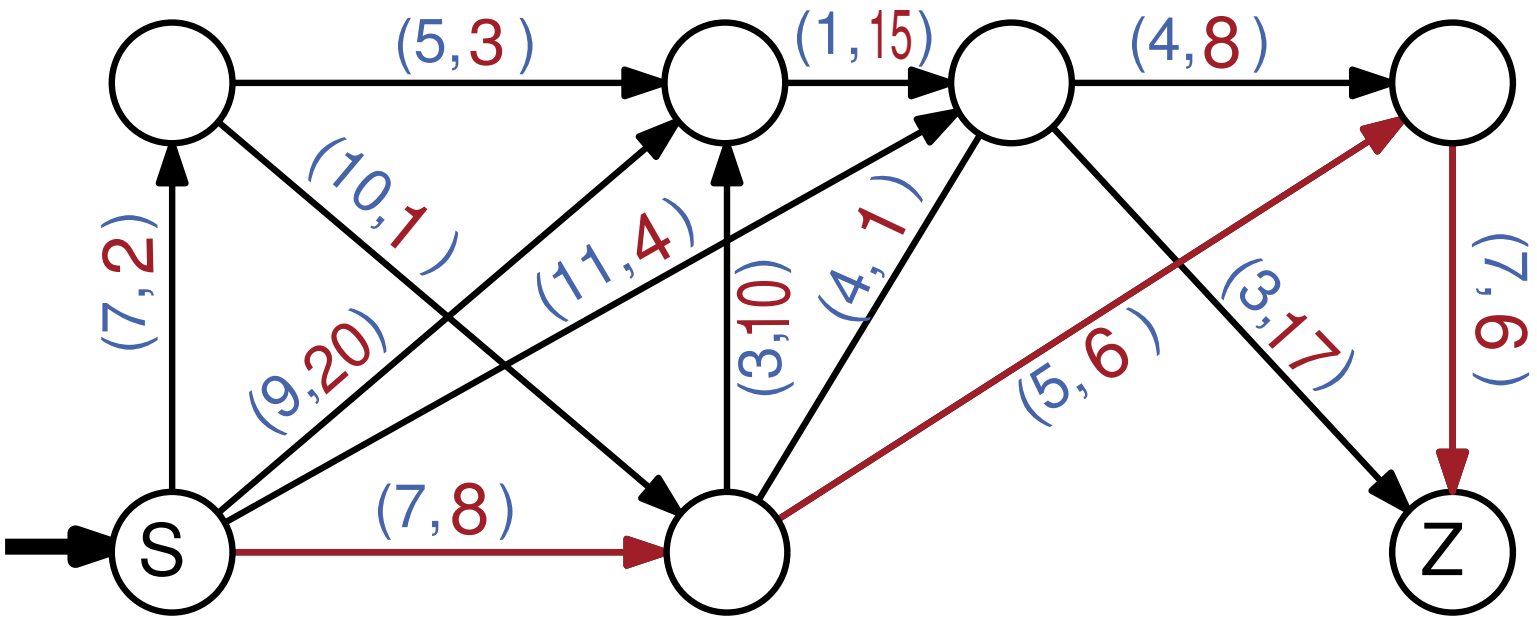
Restricted Shortest Path Problem With T = 25 — OPT = 14

Restricted Shortest Path Problem With T = 20

...

Institute of Theoretical Informatics
Algorithmics Group

Definition - (2)



Shortest Path Problem — OPT = 13

Restricted Shortest Path Problem With T = 25 — OPT = 14

Restricted Shortest Path Problem With T = 20 — OPT = 19

...

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Algorithmen

Es gibt sind zwei Pseudopolynomial Algorithmen für das problem

Aber, was ist ” Pseudopolynomial” ?

Pseduopolynomial Zeit

- Was ist polynomial zeit ?

  Generale Intuition : Zeit O(n^k) für einige k, und Eingabegröße n

Pseduopolynomial Zeit

- Was ist polynomial zeit ?

  Generale Intuition : Zeit O(n^k) für einige k, und Eingabegröße n

- Aber gilt das für den Algorithmus, der in $\mathcal{O}(n^4)$ testet ob n eine Primzahl ist? Nein!

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Zeit

- Was ist polynomial zeit ?

  Generale Intuition : Zeit O(n^k) für einige k, und Eingabegröße n

- Aber gilt das für den Algorithmus, der in $\mathcal{O}(n^4)$ testet ob n eine Primzahl ist? Nein!
- Um n darzustellen, brauchen wir O(log(n)) bits
  Sei x die Anzahl von Bits, die benötigt werden, um n darzustellen

  D.h. $n = 2^x$ → $\mathcal{O}(2^{4x})$

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Zeit

- Was ist polynomial zeit ?

  Generale Intuition : Zeit O(n^k) für einige k, und Eingabegröße n

- Aber gilt das für den Algorithmus, der in $\mathcal{O}(n^4)$ testet ob n eine Primzahl ist? Nein!
- Um n darzustellen, brauchen wir O(log(n)) bits
  Sei x die Anzahl von Bits, die benötigt werden, um n darzustellen

  D.h. $n = 2^x$ → $\mathcal{O}(2^{4x})$

Polynomial Zeit
Zeit $\mathcal{O}(n^k)$ für einige k, wobei n ist die Anzahl der Bits, die zum Ausgeben die Eingabe erforderlich sind

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Zeit

Pseudopolynomial

Die Laufzeit ist ein Polynom im numerischen Wert der Eingabe und nicht in der Anzahl der Bits, die für die Darstellung benötigt werden

Pseudopolynomial Zeit

Pseudopolynomial

Die Laufzeit ist ein Polynom im numerischen Wert der Eingabe und nicht in der Anzahl der Bits, die für die Darstellung benötigt werden

- In vielen Fällen sind Pseudopolynomialzeitalgorithmen in Ordnung
  $\rightarrow$ wenn eine obere Grenze bekannt ist

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Zeit

Pseudopolynomial

Die Laufzeit ist ein Polynom im numerischen Wert der Eingabe und nicht in der Anzahl der Bits, die für die Darstellung benötigt werden

- In vielen Fällen sind Pseudopolynomialzeitalgorithmen in Ordnung
  $\rightarrow$ wenn eine obere Grenze bekannt ist
- Beispiel : Counting-Sort mit Laufzeit $\mathcal{O}$(n+U)
  (n : Größe des Arrays, U : Die größte Zahl im Array)

Pseudopolynomial Zeit

Pseudopolynomial

Die Laufzeit ist ein Polynom im numerischen Wert der Eingabe und nicht in der Anzahl der Bits, die für die Darstellung benötigt werden

- In vielen Fällen sind Pseudopolynomialzeitalgorithmen in Ordnung
  $\rightarrow$ wenn eine obere Grenze bekannt ist
- Beispiel : Counting-Sort mit Laufzeit $\mathcal{O}$(n+U)
  (n : Größe des Arrays, U : Die größte Zahl im Array)

  U beschränken $\rightarrow$ Laufzeit von $\mathcal{O}(n)$

Institute of Theoretical Informatics
Algorithmics Group

Pseudopolynomial Algorithmen

Es gibt sind zwei Pseudopolynomial Algorithmen für das problem

Aber, was ist ” Pseudopolynomial” ?

Pseudopolynomial Algorithmen

Es gibt sind zwei Pseudopolynomial Algorithmen für das problem

Aber, was ist ” Pseudopolynomial” ?

- Dynamische Programmierung A mit Laufzeit $\mathcal{O}(|\mathbb{E}|\mathbb{T})$
- Dynamische Programmierung B mit Laufzeit $\mathcal{O}(|\mathbb{E}|\mathbb{OPT})$

wobei OPT is die Länge des kürzesten 1-n T-Wegs

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j
- Annahme : Längen und Übergangszeiten werden positive und ganze Zahlen

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j
- Annahme : Längen und Übergangszeiten werden positive und ganze Zahlen
- $c(\mathbb{S}) \rightarrow$ summe von Längen von den Kanten in Set $\mathbb{S}$.

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j
- Annahme : Längen und Übergangszeiten werden positive und ganze Zahlen
- $c(\mathbb{S}) \rightarrow$ summe von Längen von den Kanten in Set $\mathbb{S}$.
- $t(\mathbb{S}) \rightarrow$ summe von Übergangzeiten von den Kanten in Set $\mathbb{S}$.

Institute of Theoretical Informatics
Algorithmics Group

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j
- Annahme : Längen und Übergangszeiten werden positive und ganze Zahlen
- $c(\mathbb{S}) \rightarrow$ summe von Längen von den Kanten in Set $\mathbb{S}$.
- $t(\mathbb{S}) \rightarrow$ summe von Übergangzeiten von den Kanten in Set $\mathbb{S}$.
- Annahme : der Graph ist azyklisch.

Abkürzungen und Annahmen

Vor wir anfangen :

- $c_{ij} \rightarrow$ länge von Kante zwischen i und j
- $t_{ij} \rightarrow$ übergangszeit von Kante zwischen i und j
- Annahme : Längen und Übergangszeiten werden positive und ganze Zahlen
- $c(\mathbb{S}) \rightarrow$ summe von Längen von den Kanten in Set $\mathbb{S}$.
- $t(\mathbb{S}) \rightarrow$ summe von Übergangzeiten von den Kanten in Set $\mathbb{S}$.
- Annahme : der Graph ist azyklisch.
- Die Erweiterung auf allgemeine Graphen ist unkompliziert

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

Wir nehmen an, dass die Knoten so nummeriert sind, dass Kante (i, j) $\in$ E impliziert dass i $<$ j ist. Das kann in O($|E|$) gemacht werden

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1), min_{k|t_{kj} \leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

Wir nehmen an, dass die Knoten so nummeriert sind, dass Kante (i, j) $\in$ E impliziert dass i $<$ j ist. Das kann in O($|E|$) gemacht werden

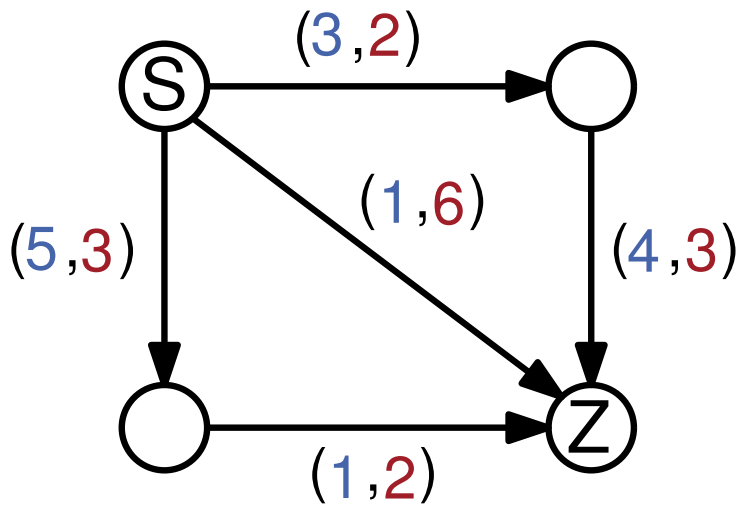
Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1), min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T



Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

Wir nehmen an, dass die Knoten so nummeriert sind, dass Kante (i, j) $\in$ E impliziert dass i $<$ j ist. Das kann in O($|E|$) gemacht werden

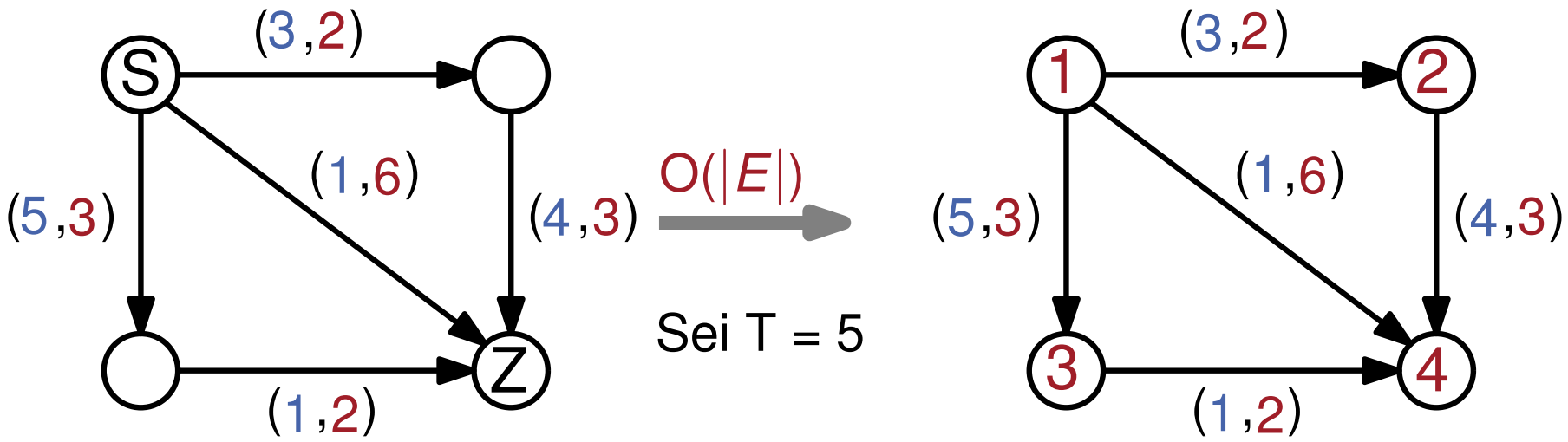
Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

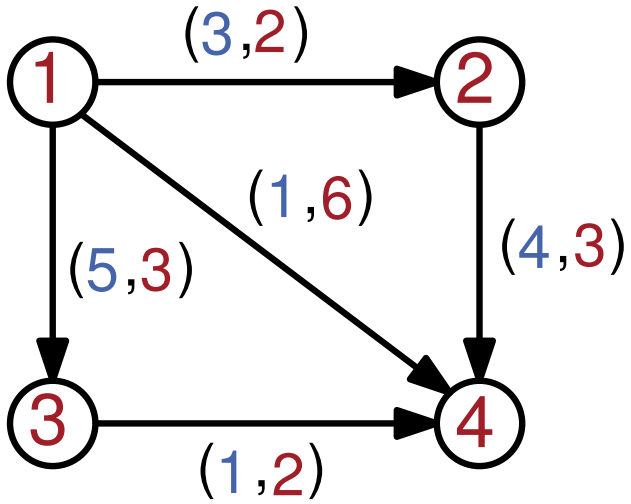
$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1), min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T



Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$



Sei T = 5

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

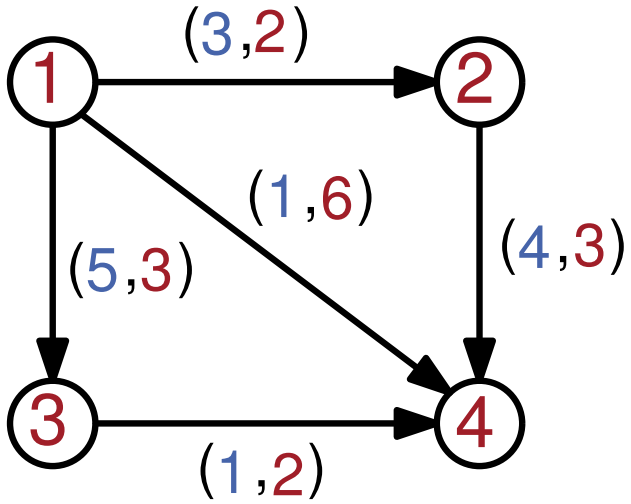
$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1),\ min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

	t=0	t=1	t=2	t=3	t=4	t=5
j=1	0	0	0	0	0	0
j=2						
j=3						
j=4						



Sei T = 5

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

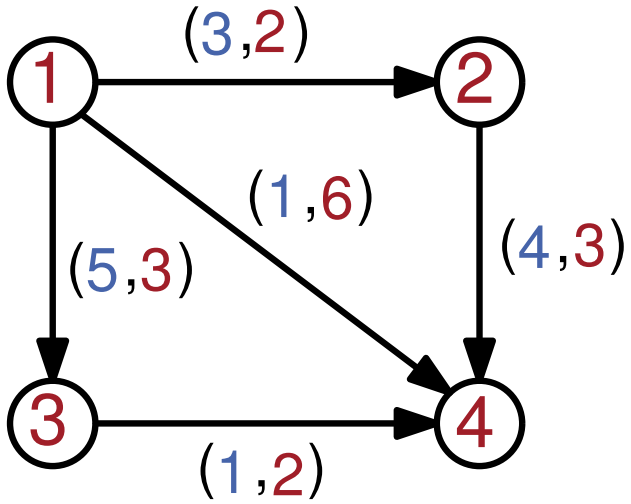
$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1),\ min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

	t=0	t=1	t=2	t=3	t=4	t=5
j=1	0	0	0	0	0	0
j=2	∞					
j=3	∞					
j=4	∞					



Sei T = 5

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

$f_j(0) = \infty,$ j = 2,..., n.

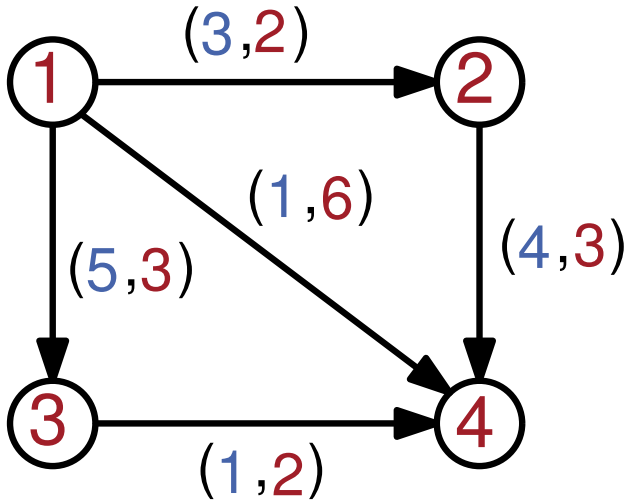
$f_j(t) = min\{f_j(t-1),\ min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

	t=0	t=1	t=2	t=3	t=4	t=5
j=1	0	0	0	0	0	0
j=2	∞	∞				
j=3	∞	∞				
j=4	∞	∞				



Sei T = 5

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

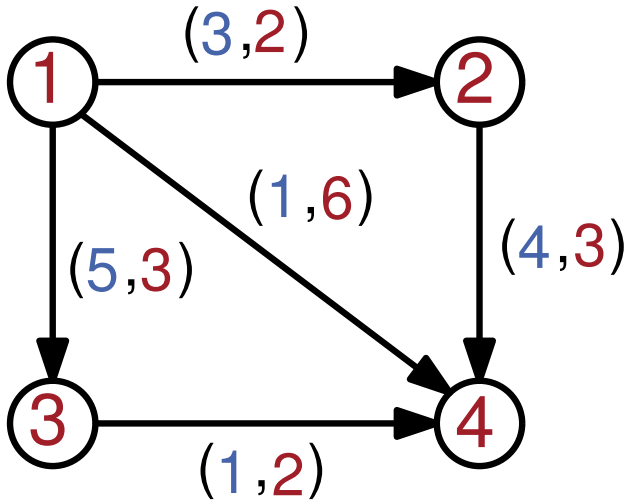
$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1),\ min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Algorithmus A Laufzeit $\mathcal{O}(|E|\mathbf{T})$

	t=0	t=1	t=2	t=3	t=4	t=5
j=1	0	0	0	0	0	0
j=2	∞	∞	3	3	3	3
j=3	∞	∞	∞	5	5	5
j=4	∞	∞	∞	∞	∞	6



Sei T = 5

Algorithm A. (Denote by $f_j(t)$ the length of a shortest 1 - j t-path)

$f_1(t) = 0,$ t = 0,..., T.

$f_j(0) = \infty,$ j = 2,..., n.

$f_j(t) = min\{f_j(t-1),\ min_{k|t_{kj}\leq t}\{f_k(t-t_{kj}) + c_{kj}\}\}$

j = 2,...n , t = 1,...,T

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\mathbf{OPT})$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c - c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\mathrm{OPT})$

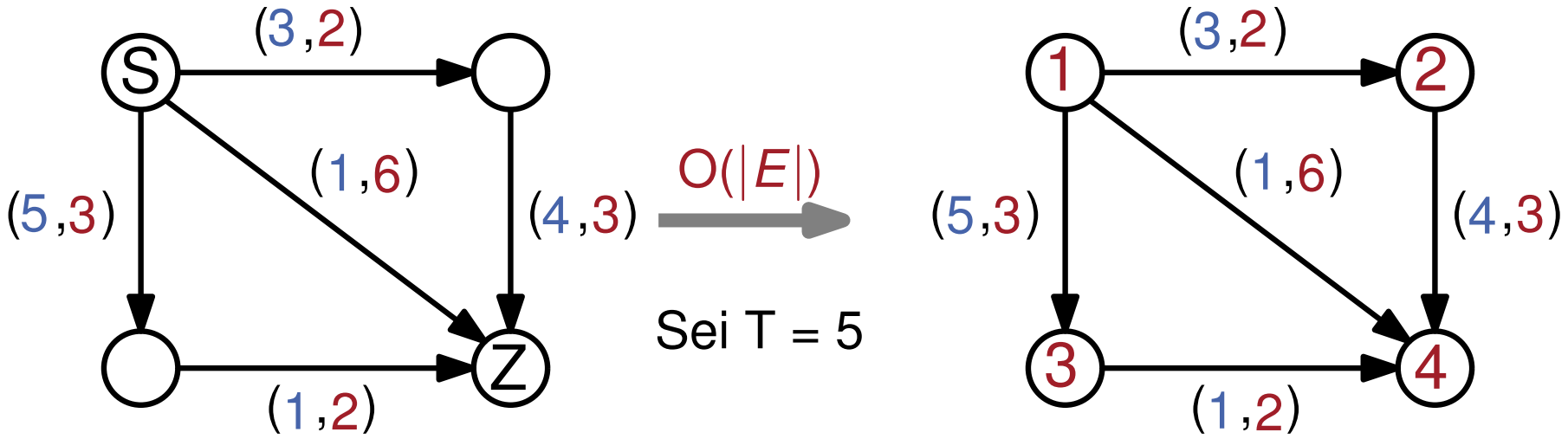
Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

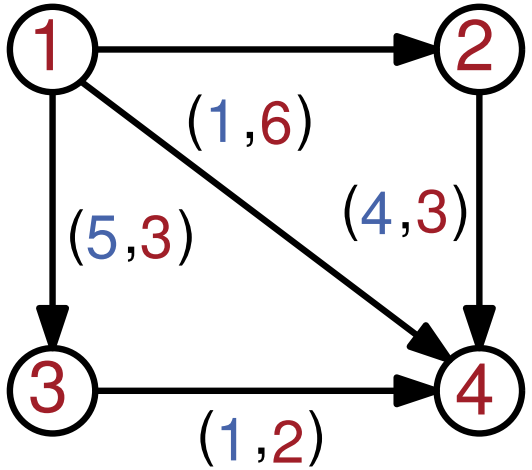
$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj}\leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT



Algorithmus B Laufzeit $\mathcal{O}(|E|\mathrm{OPT})$



Sei T = 5

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

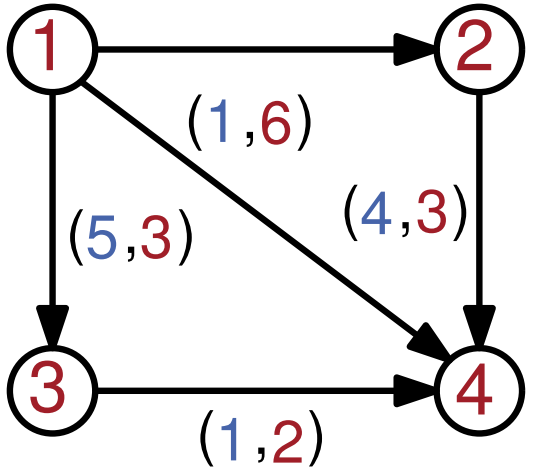
$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj}\leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0
j=1	0
j=2	∞
j=3	∞
j=4	∞



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

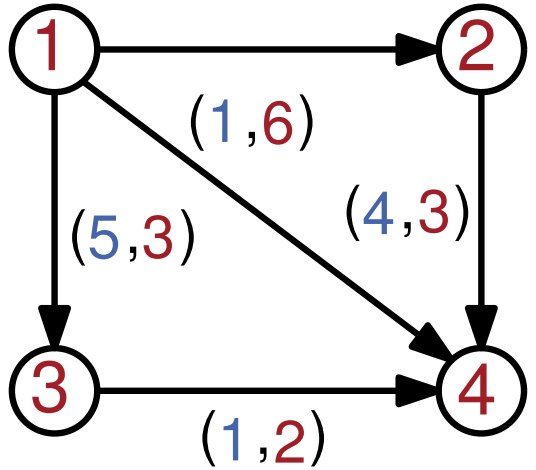
$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0	c=1
j=1	0	0
j=2	∞	∞
j=3	∞	∞
j=4	∞	6



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

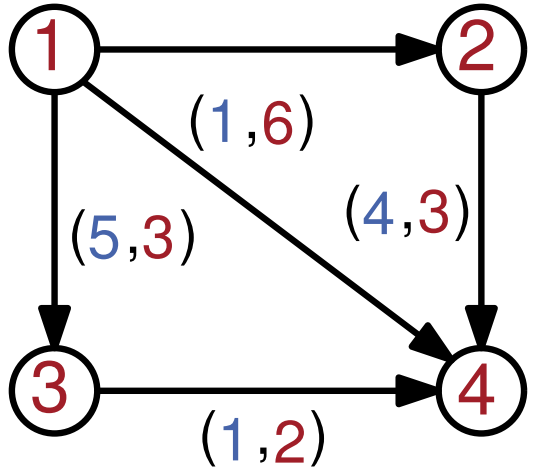
$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0	c=1	c=2
j=1	0	0	0
j=2	∞	∞	∞
j=3	∞	∞	∞
j=4	∞	6	6



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

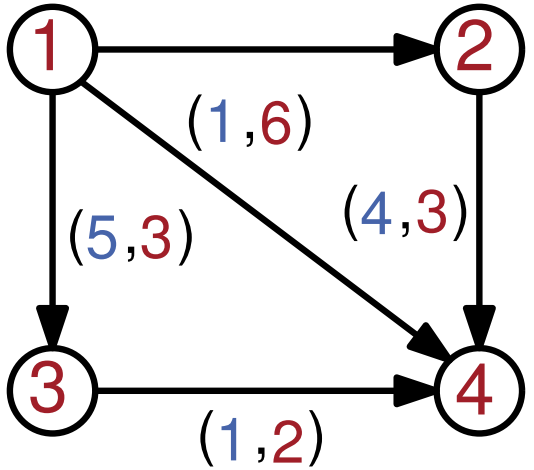
$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0	c=1	c=2	c=3
j=1	0	0	0	0
j=2	∞	∞	∞	2
j=3	∞	∞	∞	∞
j=4	∞	6	6	6



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

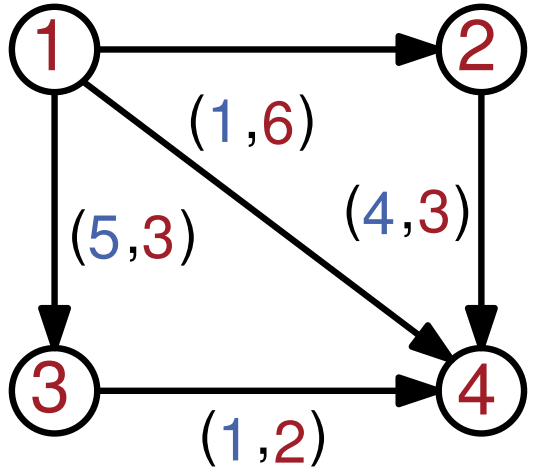
$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0	c=1	c=2	c=3	c=4
j=1	0	0	0	0	0
j=2	∞	∞	∞	2	2
j=3	∞	∞	∞	∞	∞
j=4	∞	6	6	6	6



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

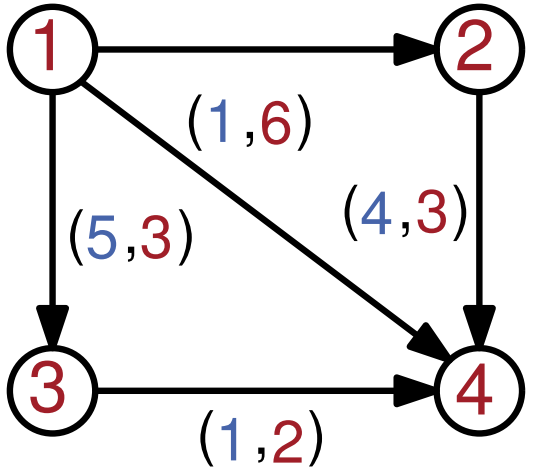
$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Algorithmus B Laufzeit $\mathcal{O}(|E|\mathrm{OPT})$

	c=0	c=1	c=2	c=3	c=4	c=5
j=1	0	0	0	0	0	0
j=2	∞	∞	∞	2	2	2
j=3	∞	∞	∞	∞	∞	3
j=4	∞	6	6	6	6	6



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

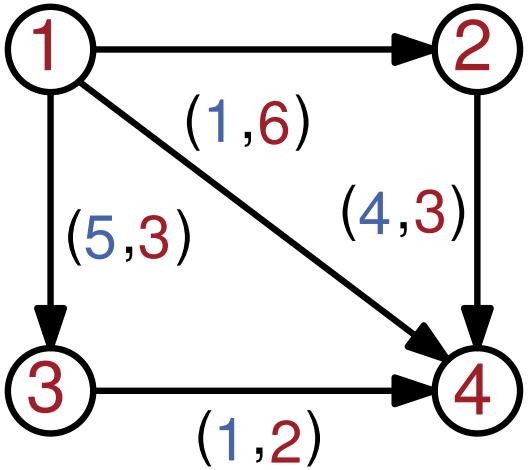
$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Institute of Theoretical Informatics
Algorithmics Group

Algorithmus B Laufzeit $\mathcal{O}(|E|\text{OPT})$

	c=0	c=1	c=2	c=3	c=4	c=5	c=6
j=1	0	0	0	0	0	0	0
j=2	∞	∞	∞	2	2	2	2
j=3	∞	∞	∞	∞	∞	3	3
j=4	∞	6	6	6	6	6	5



Sei T = 5

- OPT = min $\{c \parallel g_n(c) \leq T\}$.
  $g_n(c)$ wird von c = 1 bis zum ersten Wert von c berechnet, für den gilt $g_n(c) \leq T$

Algorithm B. (Denote by $g_j(c)$ the time of the quickest 1-j t-path with length is at most c)

$g_1(c) = 0$, c = 0,..., OPT.

$g_j(0) = \infty$, j = 2,..., n.

$g_j(c) = min\{g_j(c-1), min_{k|c_{kj} \leq c}\{g_k(c-c_{kj}) + t_{kj}\}\}$

j = 2,...n , c = 1,....OPT

Gliederung

- Definition
- Zwei exakte Pseudopolyomial-Algorithmen vorstellen
- ”TEST(V)” Hilfsmethode für das erste FPTAS
- Erstes FPTAS (Rounding) von Komplexität :

  $\mathcal{O}(\log(\log(B)) \cdot (|E|(n/\varepsilon) + \log(\log(B))$
- ”Partition” Hilfsmethode für das zweite FPTAS
- Zweites FPTAS (Interval Partitioning) von Komplexität :

  $\mathcal{O}(|E|(n^2/\varepsilon) \cdot \log(n/\varepsilon))$

Institute of Theoretical Informatics
Algorithmics Group

Test Hilfsmethode

Hauptidee :

- Statt direct Opt-wert zu berechnen,
  wir haben eine Test-Methode die entscheidet, ob Opt $\geq$ V für gegeben V

Institute of Theoretical Informatics
Algorithmics Group

Test Hilfsmethode

Hauptidee :

- Statt direct Opt-wert zu berechnen,
  wir haben eine Test-Methode die entscheidet, ob Opt $\geq$ V für gegeben V
- Wenn diese Methode polynomial ist, dann kann sie auf andere erweitert, die Exact wert von OPT berechnet.

  $\rightarrow$ Mittels Binärsuche auf $\{ 0, \ldots , UB \}$

Institute of Theoretical Informatics
Algorithmics Group

Test Hilfsmethode

Hauptidee :

- Statt direct Opt-wert zu berechnen,
  wir haben eine Test-Methode die entscheidet, ob Opt $\geq$ V für gegeben V
- Wenn diese Methode polynomial ist, dann kann sie auf andere erweitert, die Exact wert von OPT berechnet.

$\rightarrow$ Mittels Binärsuche auf $\{ 0, \ldots , UB \}$

Wir stellen nun vor, wie man einen
Polynom-Zeit-ϵApproximationstest erstellt, mit $0 < \epsilon < 1$

Institute of Theoretical Informatics
Algorithmics Group

Test Hilfsmethode

Hauptidee :

- Statt direct Opt-wert zu berechnen,
  wir haben eine Test-Methode die entscheidet, ob Opt $\geq$ V für gegeben V
- Wenn diese Methode polynomial ist, dann kann sie auf andere erweitert, die Exact wert von OPT berechnet.

$\rightarrow$ Mittels Binärsuche auf $\{ 0, \ldots , UB \}$

Wir stellen nun vor, wie man einen
Polynom-Zeit-ϵApproximationstest erstellt, mit $0 < \epsilon < 1$

Wenn Test(V) Ja zurückgibt $\longrightarrow$ OPT $\geq$ V

Wenn Test(V) Nein zurückgibt $\longrightarrow$ OPT $<$ V(1+ ϵ)

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure

Hauptidee :

- Rundung $\longrightarrow$ $c_{ij}' = \lfloor c_{ij} / (\frac{V\epsilon}{n-1}) \rfloor (\frac{V\epsilon}{n-1})$

  Dies verringert höchstens jede Kantenlänge um $(\frac{V\epsilon}{n-1})$ und jede Pfadlänge um $(V\epsilon)$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure

Hauptidee :

- Rundung $\longrightarrow$ $c_{ij}' = \lfloor c_{ij} / (\frac{V\epsilon}{n-1}) \rfloor (\frac{V\epsilon}{n-1})$

  Dies verringert höchstens jede Kantenlänge um $(\frac{V\epsilon}{n-1})$ und jede Pfadlänge um $(V\epsilon)$

- Algorithmus B auf die skalierte-Kantenlängen $\lfloor c_{ij} / ((V\epsilon)/(n-1)) \rfloor$

  Wir berechnen $g_j(c)$ für jedes c bis :

  $g_n(c) \leq T$ für c = c' $< (n-1)/\epsilon$ ODER c $\geq (n-1)/\epsilon$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure

Hauptidee :

- Rundung $\longrightarrow$ $c_{ij}' = \lfloor c_{ij} / (\frac{V\epsilon}{n-1}) \rfloor (\frac{V\epsilon}{n-1})$

  Dies verringert höchstens jede Kantenlänge um $(\frac{V\epsilon}{n-1})$ und jede Pfadlänge um $(V\epsilon)$

- Algorithmus B auf die skalierte-Kantenlängen $\lfloor c_{ij} / ((V\epsilon)/(n-1)) \rfloor$

  Wir berechnen $g_j(c)$ für jedes c bis :

  $g_n(c) \leq T$ für c = c' $< (n-1)/\epsilon$ ODER c $\geq (n-1)/\epsilon$

$$c' < \frac{n-1}{\epsilon} \Rightarrow c' \frac{V\epsilon}{n-1} < \frac{n-1}{\epsilon} \frac{V\epsilon}{n-1}$$

$$\Rightarrow c' \frac{V\epsilon}{n-1} + V\epsilon < V + V\epsilon = V(1+\epsilon)$$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure

Hauptidee :

- Rundung $\longrightarrow$ $c_{ij}' = \lfloor c_{ij} / (\frac{V\epsilon}{n-1}) \rfloor (\frac{V\epsilon}{n-1})$

  Dies verringert höchstens jede Kantenlänge um $(\frac{V\epsilon}{n-1})$ und jede Pfadlänge um $(V\epsilon)$

- Algorithmus B auf die skalierte-Kantenlängen $\lfloor c_{ij} / ((V\epsilon)/(n-1)) \rfloor$

  Wir berechnen $g_j(c)$ für jedes c bis :

  $g_n(c) \leq T$ für c = c' $< (n-1)/\epsilon$ ODER c $\geq (n-1)/\epsilon$

2 .Fall : Jeder T-Pfad hat c" $\geq \frac{n-1}{\epsilon} \Rightarrow c" \frac{V\epsilon}{n-1} \geq \frac{n-1}{\epsilon} \frac{V\epsilon}{n-1}$

$\Rightarrow$ Jeder T-Pfad hat $c \geq V$, damit ist $OPT \geq V$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure

Hauptidee :

- Rundung $\longrightarrow$ $c_{ij}' = \lfloor c_{ij} / (\frac{V\epsilon}{n-1}) \rfloor (\frac{V\epsilon}{n-1})$

  Dies verringert höchstens jede Kantenlänge um $(\frac{V\epsilon}{n-1})$ und jede Pfadlänge um $(V\epsilon)$

- Algorithmus B auf die skalierte-Kantenlängen $\lfloor c_{ij} / ((V\epsilon)/(n-1)) \rfloor$

  Wir berechnen $g_j(c)$ für jedes c bis :

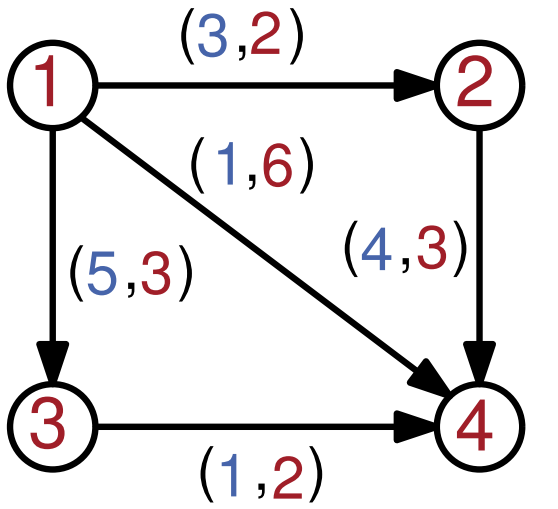
  $g_n(c) \leq T$ für c = c' $< (n-1)/\epsilon$ ODER $c \geq (n-1)/\epsilon$

1.Fall : T-Pfad von länge maximal $V\epsilon/(n-1)c' + V\epsilon < V(1+\epsilon)$ ist gefunden	2 .Fall : Jeder T-Pfad hat $c \geq V$. Damit ist $OPT \geq V$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

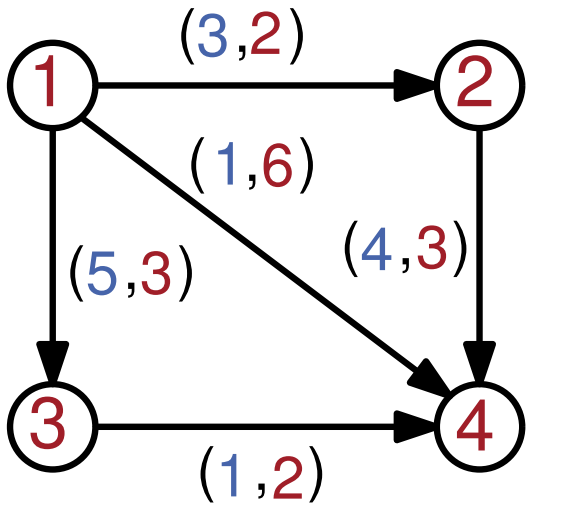
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



Sei V = 4, ϵ = 0.5

Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



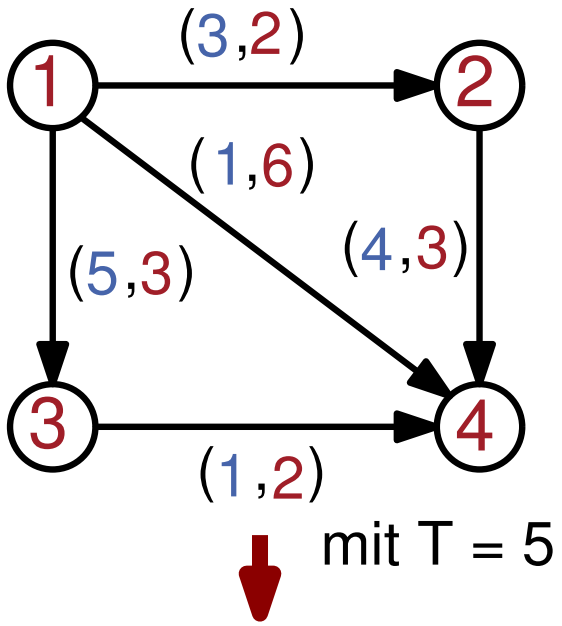
mit T = 5

Sei V = 4, ϵ = 0.5 c = 0

Institute of Theoretical Informatics
Algorithmics Group

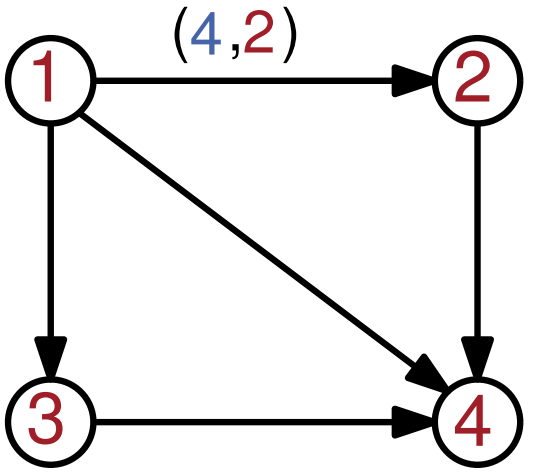
Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



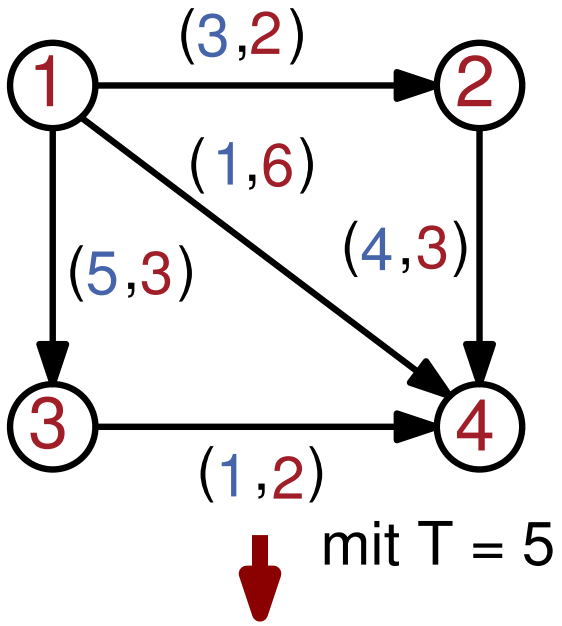
Sei V = 4, ϵ = 0.5 c = 0

$$c_{12} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 3(4-1)/(4*0.5) \rfloor = 4$$



Test(V) Procedure - Beispiel

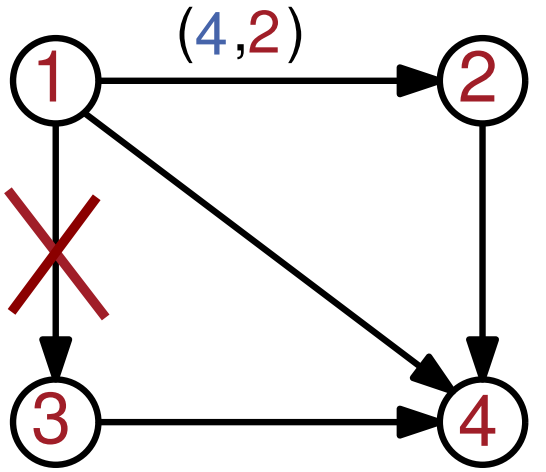
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



Sei V = 4, ϵ = 0.5 c = 0

$c_{12} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 3(4-1)/(4*0.5) \rfloor = 4$

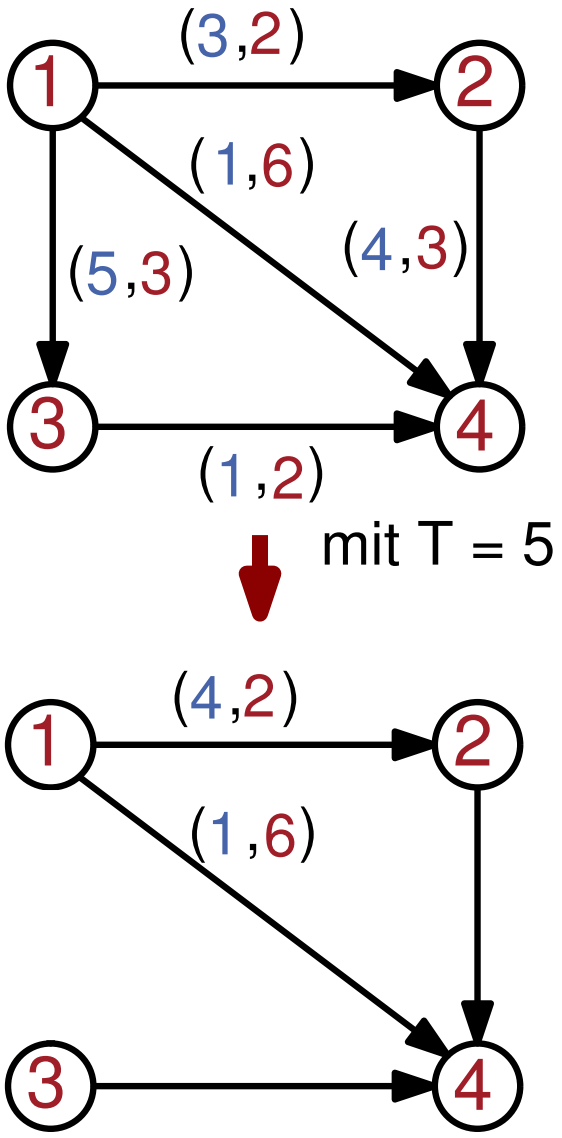
c_{13} wird aus dem Set entfernt



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



Sei V = 4, ϵ = 0.5 c = 0

$c_{12} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 3(4-1)/(4*0.5) \rfloor = 4$

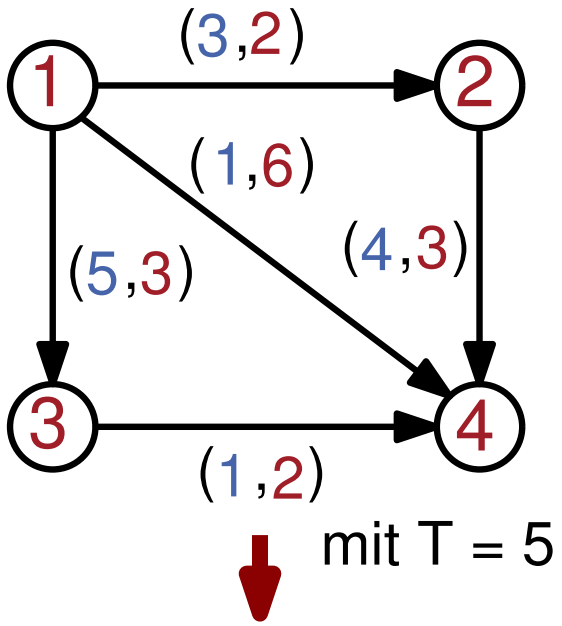
c_{13} wird aus dem Set entfernt

$c_{14} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 1(4-1)/(4*0.5) \rfloor = 1$

Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



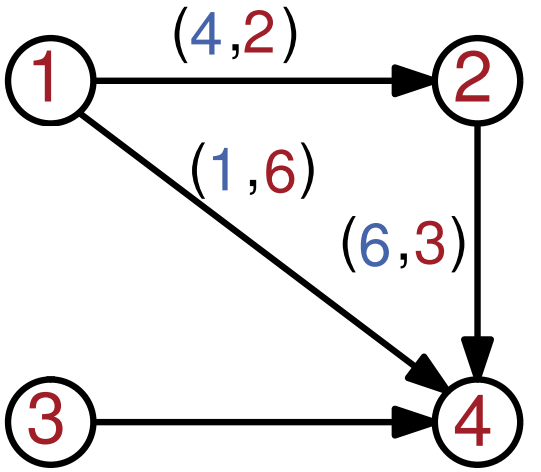
Sei V = 4, ϵ = 0.5 c = 0

$c_{12} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 3(4-1)/(4*0.5) \rfloor = 4$

c_{13} wird aus dem Set entfernt

$c_{14} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 1(4-1)/(4*0.5) \rfloor = 1$

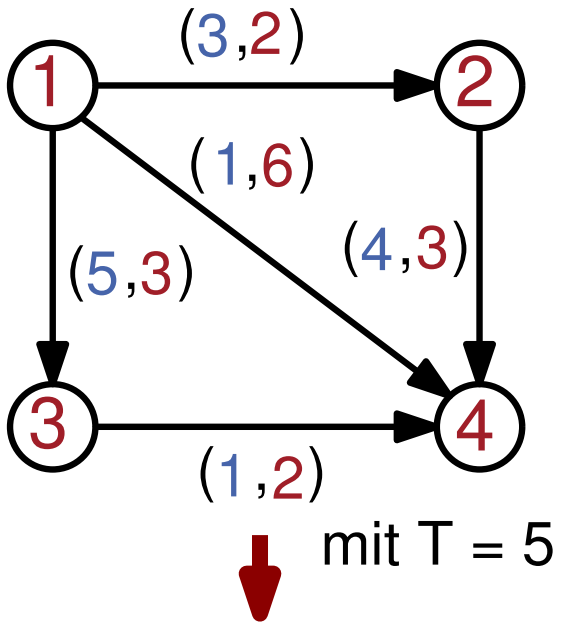
$c_{24} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 4(4-1)/(4*0.5) \rfloor = 6$



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



Sei V = 4, ϵ = 0.5 c = 0

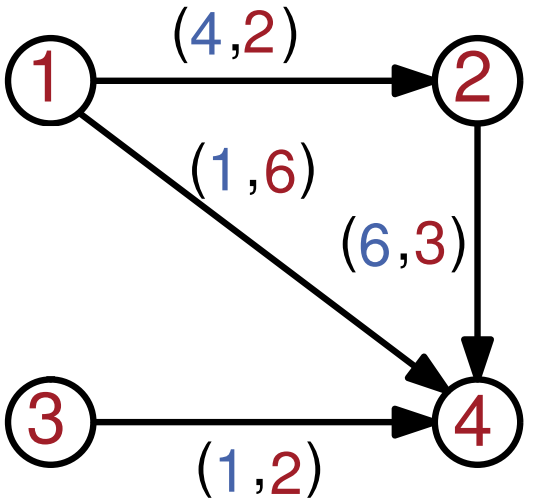
$c_{12} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 3(4-1)/(4*0.5) \rfloor = 4$

c_{13} wird aus dem Set entfernt

$c_{14} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 1(4-1)/(4*0.5) \rfloor = 1$

$c_{24} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 4(4-1)/(4*0.5) \rfloor = 6$

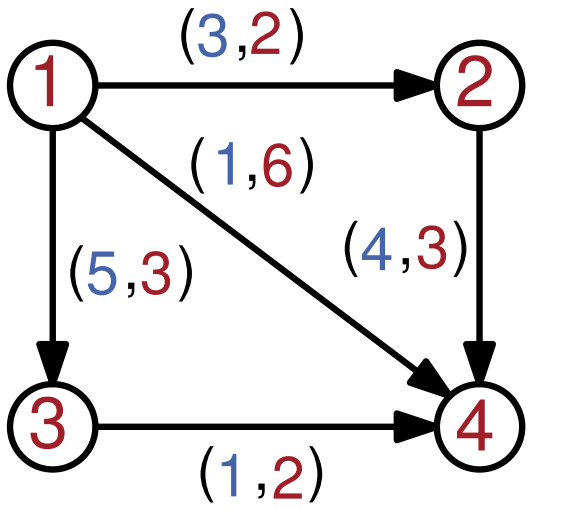
$c_{34} = \lfloor c_{12}(n-1)/V\epsilon \rfloor = \lfloor 1(4-1)/(4*0.5) \rfloor = 1$



Institute of Theoretical Informatics
Algorithmics Group

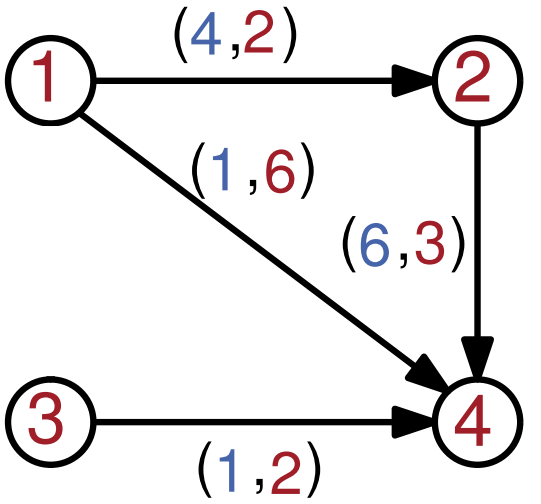
Test(V) Procedure - Beispiel

1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

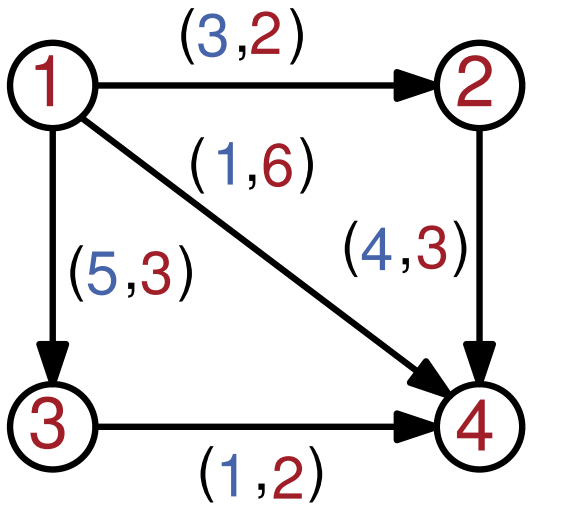
Sei V = 4, ϵ = 0.5 c = 0



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

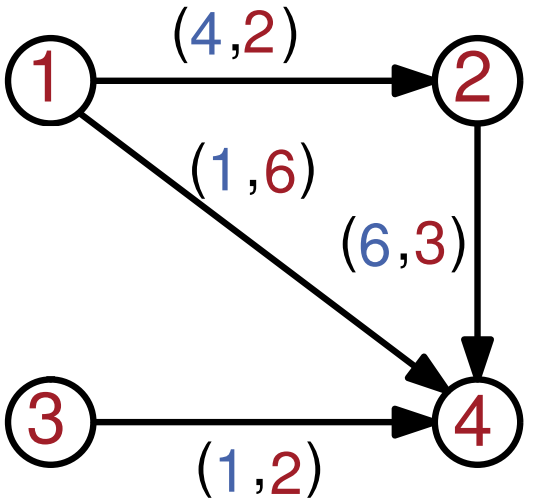
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 0

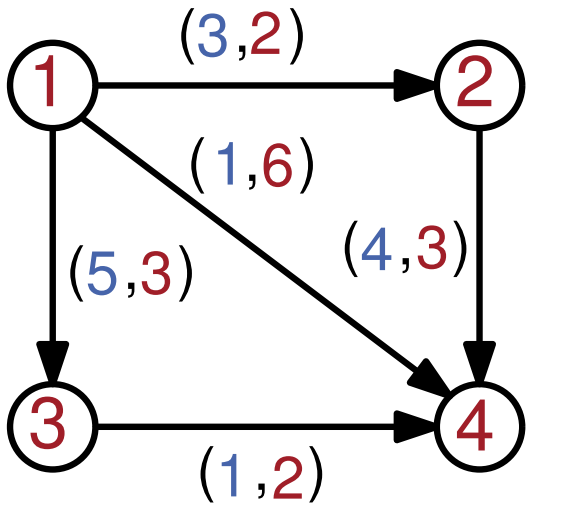
	c=0
j=1	0
j=2	∞
j=3	∞
j=4	∞



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

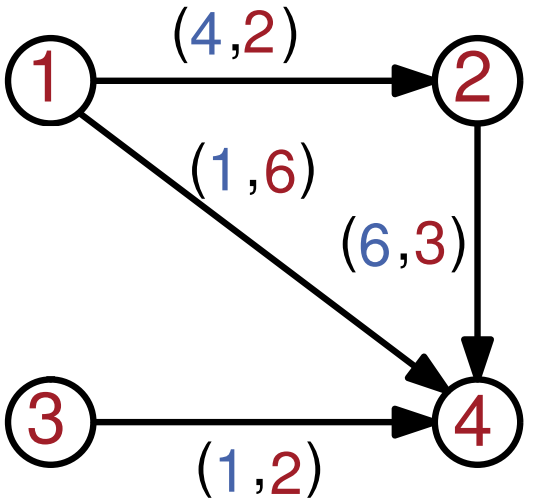
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 1

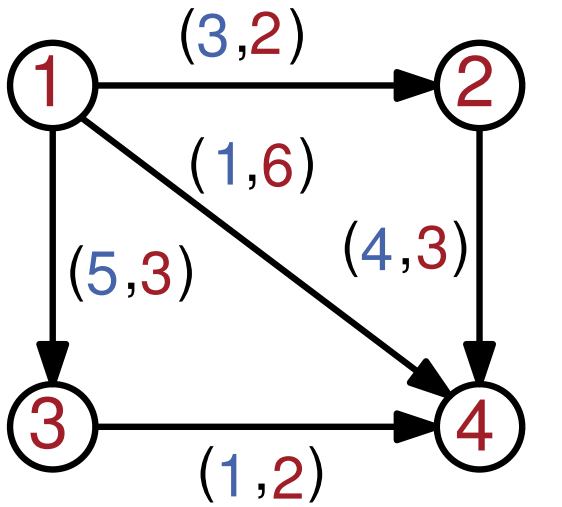
	c=0	c=1
j=1	0	0
j=2	∞	∞
j=3	∞	∞
j=4	∞	6



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

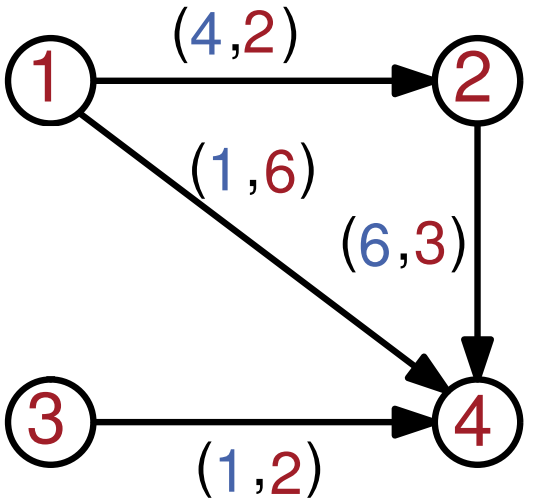
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 2

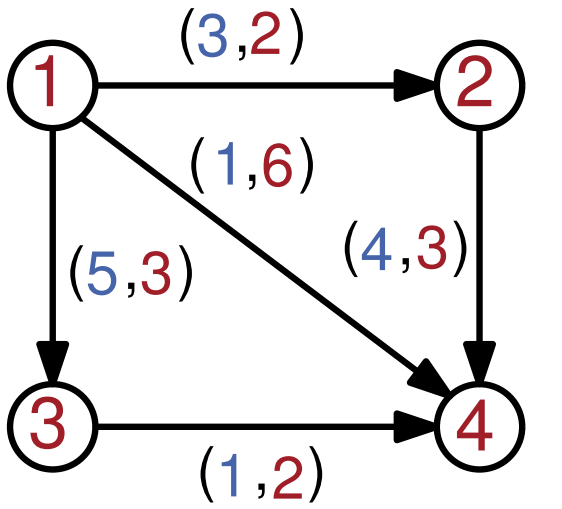
	c=0	c=1	c=2
j=1	0	0	0
j=2	∞	∞	∞
j=3	∞	∞	∞
j=4	∞	6	6



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

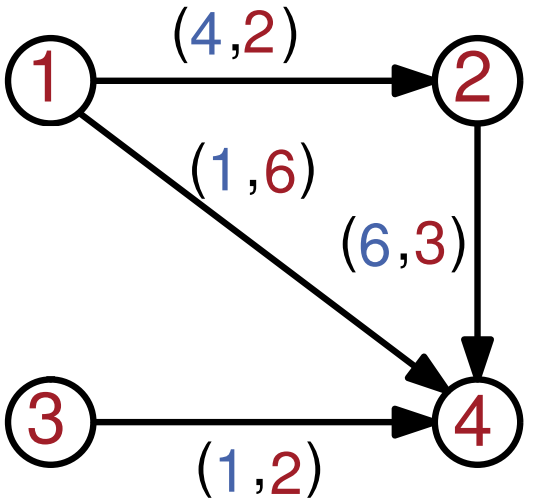
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

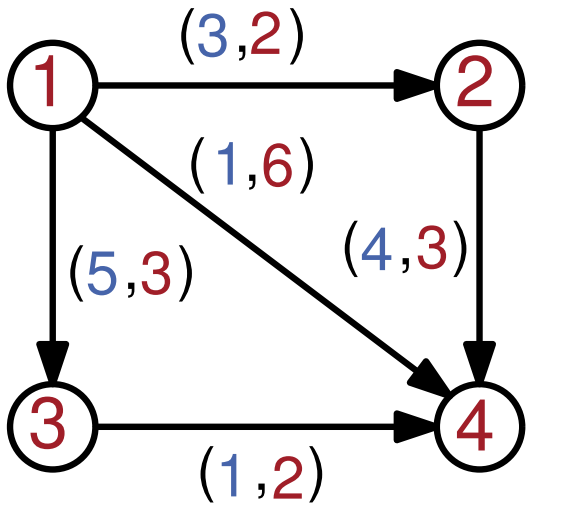
Sei V = 4, ϵ = 0.5 c = 3

	c=0	c=1	c=2	c=3
j=1	0	0	0	0
j=2	∞	∞	∞	2
j=3	∞	∞	∞	∞
j=4	∞	6	6	6



Test(V) Procedure - Beispiel

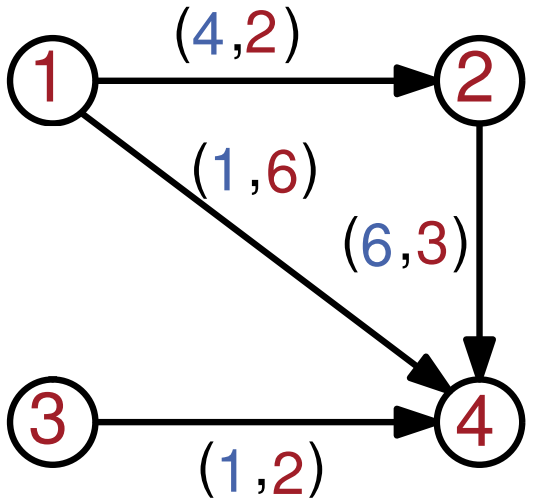
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 4

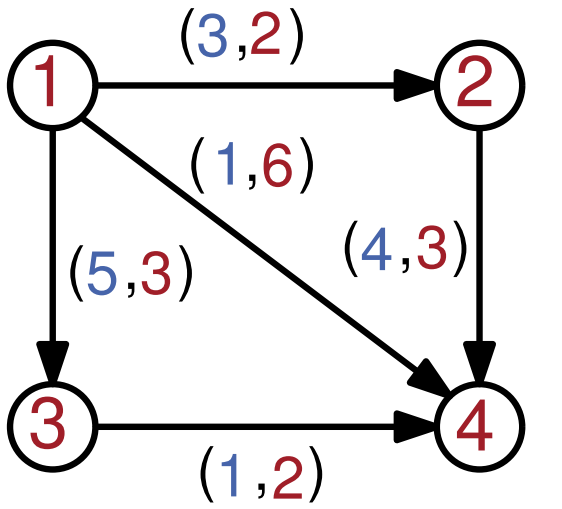
	c=0	c=1	c=2	c=3	c=4
j=1	0	0	0	0	0
j=2	∞	∞	∞	2	2
j=3	∞	∞	∞	∞	∞
j=4	∞	6	6	6	6



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

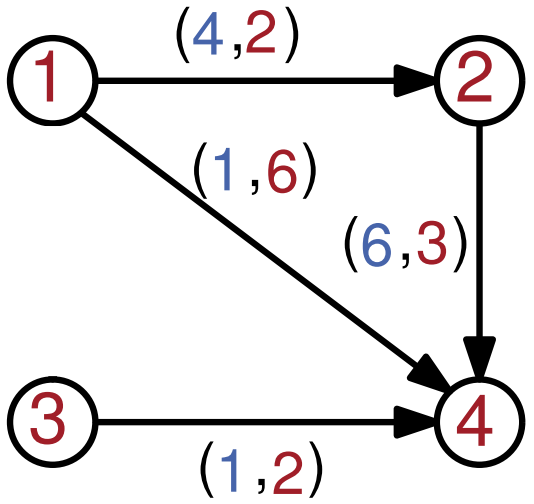
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 5

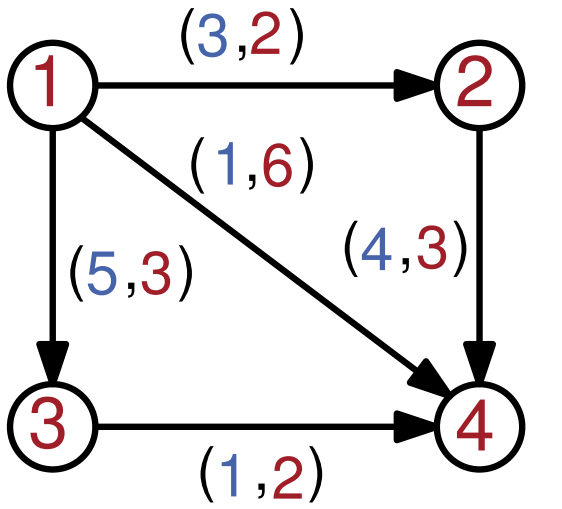
	c=0	c=1	c=2	c=3	c=4	c=5
j=1	0	0	0	0	0	0
j=2	∞	∞	∞	2	2	2
j=3	∞	∞	∞	∞	∞	3
j=4	∞	6	6	6	6	6



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Beispiel

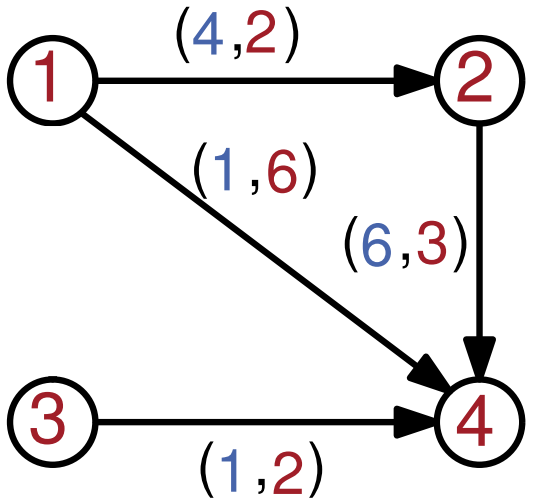
1. Set c $\leftarrow$ 0
   For all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** $E \leftarrow E \setminus \{(i,j)\}$
   Else, then set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output**NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.



mit T = 5

Sei V = 4, ϵ = 0.5 c = 6

	c=0	c=1	c=2	c=3	c=4	c=5
j=1	0	0	0	0	0	0
j=2	∞	∞	∞	2	2	2
j=3	∞	∞	∞	∞	∞	3
j=4	∞	6	6	6	6	6



Institute of Theoretical Informatics
Algorithmics Group

Test(V) Procedure - Analyse

Procedure TEST(V).

1. Set c $\leftarrow$ 0
   for all (i,j) $\in$ E :
   If $c_{ij} > V$, **then set** E$\leftarrow$ E $\setminus\{(i,j)\}$
   Else, set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/V\epsilon \rfloor$
2. **If** c $\geq (n-1)/\epsilon$ **then output** YES.
   Else use Algorithm B to compute $g_j(c)$ for j = 2,... n.
   If $g_n(c) \leq$ T **then output** NO.
   Else set c $\leftarrow$ c + 1 and repeat Step 2.

Analyse :

- schritte 1 (Rundung) : $O(|E| \log(n/\epsilon))$
  von oben durch V begrenzt
- Schritte 2 : $O(|E|n/\epsilon)$
  $O(n/\epsilon)$ Iterationen von Algorithmus B

$O(|E|n/\epsilon)$ ist die Komplexität des gesamten Verfahrens

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling

- Um OPT zu approximieren

  $\rightarrow$ berechenbare obere und untere Grenzen.

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling

- Um OPT zu approximieren

  $\rightarrow$ berechenbare obere und untere Grenzen.

- UB kann auf die summe der (n-1) längsten Kanten gesetzt werden.
- LB kann einfach auf 1 gesetzt werden.

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling

- Um OPT zu approximieren

  $\rightarrow$ berechenbare obere und untere Grenzen.

- UB kann auf die summe der (n-1) längsten Kanten gesetzt werden.
- LB kann einfach auf 1 gesetzt werden.

- Falls UB $\leq$ (1+ϵ)LB dann ist UB eine ϵ-Approximation von OPT
- Falls UB $>$ (1+ϵ)LB dann sei V existiert damit LB $<$ V $<$ UB$(1+\epsilon)^{-1}$
  - Hauptidee vom Algorithmus :

    1- Das Verhältnis UB/LB zu reduzieren.

    $\rightarrow$ Test(V) benuzten

    2- Algorithmus B auf skalierte Kantenlängen anwenden

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling - Pseudo Code

Rounding Algorithm.

0. Set LB and UB to their initial values.

(For example, set LB = 1, UB = sum of (n-1) longest edge-lengths.)

1. If UB $\leq$ 2LB go to Step 2.

   Let V = $(LB.UB)^{1/2}$.

   If TEST(V) = YES, set LB = V.

   If TEST(V) = NO, set UB = $V(1+\epsilon)$.

   Repeat Step 1.

2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB \rfloor$

   Apply Algorithm B to compute an optimal T-path.

   Output this path.

Erste FPAS : Rounding and Scaling - Pseudo Code

Rounding Algorithm.
0. Set LB and UB to their initial values.
(For example, set LB = 1, UB = sum of (n-1) longest edge-lengths.)
1. If UB $\leq$ 2LB go to Step 2.
   Let V = $(LB.UB)^{1/2}$.
   If TEST(V) = YES, set LB = V.
   If TEST(V) = NO, set UB = $V(1+\epsilon)$.
   Repeat Step 1.
2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB \rfloor$
   Apply Algorithm B to compute an optimal T-path.
   Output this path.

Analyse :

- V berechnen $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$

Erste FPAS : Rounding and Scaling - Pseudo Code

Rounding Algorithm.

0. Set LB and UB to their initial values.
(For example, set LB = 1, UB = sum of (n-1) longest edge-lengths.)
1. If UB $\leq$ 2LB go to Step 2.
   Let V = $(LB.UB)^{1/2}$.
   If TEST(V) = YES, set LB = V.
   If TEST(V) = NO, set UB = $V(1+\epsilon)$.
   Repeat Step 1.
2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB \rfloor$
   Apply Algorithm B to compute an optimal T-path.
   Output this path.

Analyse :

- V berechnen $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$
- Test(V) $\rightarrow \mathcal{O}(|E|n/\epsilon)$

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling - Pseudo Code

Rounding Algorithm.
0. Set LB and UB to their initial values.
(For example, set LB = 1, UB = sum of (n-1) longest edge-lengths.)
1. If UB $\leq$ 2LB go to Step 2.
   Let V = $(LB.UB)^{1/2}$.
   If TEST(V) = YES, set LB = V.
   If TEST(V) = NO, set UB = $V(1+\epsilon)$.
   Repeat Step 1.
2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB \rfloor$
   Apply Algorithm B to compute an optimal T-path.
   Output this path.

Analyse :

- V berechnen $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$
- Test(V) $\rightarrow \mathcal{O}(|E|n/\epsilon)$
- Um die Ratio unter 2 zu reduzieren $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$ Tests

Institute of Theoretical Informatics
Algorithmics Group

Erste FPAS : Rounding and Scaling - Pseudo Code

Rounding Algorithm.
0. Set LB and UB to their initial values.
(For example, set LB = 1, UB = sum of (n-1) longest edge-lengths.)
1. If UB $\leq$ 2LB go to Step 2.
   Let V = $(LB.UB)^{1/2}$.
   If TEST(V) = YES, set LB = V.
   If TEST(V) = NO, set UB = $V(1+\epsilon)$.
   Repeat Step 1.
2. Set $c_{ij} \leftarrow \lfloor c_{ij}(n-1)/\epsilon LB \rfloor$
   Apply Algorithm B to compute an optimal T-path.
   Output this path.

Analyse :

- V berechnen $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$
- Test(V) $\rightarrow \mathcal{O}(|E|n/\epsilon)$
- Um die Ratio unter 2 zu reduzieren $\rightarrow \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)$ Tests

$$\text{Laufzeit} : \mathcal{O}\left(\log\left(\log(\frac{UB}{LB})\right)\right)\left(|E|n/\epsilon + \log\left(\log(\frac{UB}{LB})\right)\right)$$

Institute of Theoretical Informatics
Algorithmics Group

Gliederung

- Definition
- Zwei exakte Pseudopolyomial-Algorithmen vorstellen
- ”TEST(V)” Hilfsmethode für das erste FPTAS
- Erstes FPTAS (Rounding) von Komplexität :

  $\mathcal{O}(\log(\log(B)) \cdot (|E|(n/\varepsilon) + \log(\log(B))$
- ”Partition” Hilfsmethode für das zweite FPTAS
- Zweites FPTAS (Interval Partitioning) von Komplexität :

  $\mathcal{O}(|E|(n^2/\varepsilon) \cdot \log(n/\varepsilon))$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$

  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :

  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$

  und $p_i \in R_{k+1}$ nur wenn $p_i > X$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$

  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :

  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$

  und $p_i \in R_{k+1}$ nur wenn $p_i > X$
- Das kann einfach in $O(m \log(k))$ Operationen durchgeführt

  $\rightarrow p_i \in R_j$ für j = min $\{$ k+1, $\lceil kp_1/X \rceil$ $\}$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$

  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :

  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$

  und $p_i \in R_{k+1}$ nur wenn $p_i > X$

- Das kann einfach in $O(m \log(k))$ Operationen durchgeführt

  $\rightarrow p_i \in R_j$ für j = min { k+1, $\lceil kp_1/X \rceil$ }

Beispiel :

Sie Q = { 1 , 3 , 4 , 6 , 8 , 10 } , X = 7 und K = 3

$p_1 = 1$, j = min$\{4, \lceil 3 * 1/7 \rceil\}$ = min$\{4,1\}$ = 1 $\rightarrow p_1 \in R_1$

$p_2 = 3$, j = min$\{4, \lceil 3 * 3/7 \rceil\}$ = min$\{4,2\}$ = 2 $\rightarrow p_2 \in R_2$

$p_3 = 4$, j = min$\{4, \lceil 3 * 4/7 \rceil\}$ = min$\{4,2\}$ = 2 $\rightarrow p_3 \in R_2$

$p_4 = 6$, j = min$\{4, \lceil 3 * 6/7 \rceil\}$ = min$\{4,3\}$ = 3 $\rightarrow p_4 \in R_3$

$p_5 = 8$, j = min$\{4, \lceil 3 * 8/7 \rceil\}$ = min$\{4,4\}$ = 4 $\rightarrow p_5 \in R_4$

$p_6 = 10$, j = min$\{4, \lceil 3 * 10/7 \rceil\}$ = min$\{4,5\}$ = 4 $\rightarrow p_6 \in R_4$

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$

  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :

  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$

  und $p_i \in R_{k+1}$ nur wenn $p_i > X$
- Das kann einfach in $O(m \log(k))$ Operationen durchgeführt

  $\rightarrow p_i \in R_j$ für j = min $\{$ k+1, $\lceil kp_1/X \rceil$ $\}$
- Diese Art von Verteilung brauchen wir in unser FPTAS

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$

  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :

  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$

  und $p_i \in R_{k+1}$ nur wenn $p_i > X$

- Das kann einfach in $O(m \log(k))$ Operationen durchgeführt

  $\rightarrow p_i \in R_j$ für j = min $\{$ k+1, $\lceil kp_1/X \rceil$ $\}$

- Diese Art von Verteilung brauchen wir in unser FPTAS

  Für jede $p_i, p_m \in R_j$ ist $|p_i - p_m| \leq \frac{X}{k}$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

- Idee : Gegeben Set Q = $\{p_1, ..., p_m\}$, X und K $\in N_+$
  Ziel ist Verteilung von Q in Subsets $R_1, ..., R_{k+1}$ sodass :
  $p_i \in R_j$ nur wenn $X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$
  und $p_i \in R_{k+1}$ nur wenn $p_i > X$
- Das kann einfach in $O(m \log(k))$ Operationen durchgeführt
  $\rightarrow p_i \in R_j$ für j = min $\{$ k+1, $\lceil kp_1/X \rceil$ $\}$
- Diese Art von Verteilung brauchen wir in unser FPTAS

  Für jede $p_i, p_m \in R_j$ ist $|p_i - p_m| \leq \frac{X}{k}$
- Angenommen dass X nicht bekannt ist, aber ein Test ist vorhanden, der entscheidet ob X $\geq$ V oder X $<$ V für beliebig V

  $\rightarrow$ Das führt zu der Partition Hilfsmethode

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = k p_i / j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots , \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = k p_i / j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots , \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

$$P_{i,(j-1)} > \pi_g \geq P_{ij}$$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = k p_i / j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots , \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

$$P_{i,(j-1)} > \pi_g \geq P_{ij} \longrightarrow P_{i,(j-1)} \geq \pi_{g+1} \geq X \geq P_{ij}$$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

$$P_{i,(j-1)} > \pi_g \geq P_{ij} \longrightarrow P_{i,(j-1)} \geq \pi_{g+1} \geq X \geq P_{ij}$$

$$\longrightarrow K\frac{P_i}{(j-1)} \geq X \geq K\frac{P_i}{(j)}$$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = k p_i / j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots , \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

$$P_{i,(j-1)} > \pi_g \geq P_{ij} \longrightarrow P_{i,(j-1)} \geq \pi_{g+1} \geq X \geq P_{ij}$$

$$\longrightarrow K\frac{P_i}{(j-1)} \geq X \geq K\frac{P_i}{(j)} \longrightarrow X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$$

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g , \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Warum funktioniert der Algorithmus ?

$$P_{i,(j-1)} > \pi_g \geq P_{ij} \longrightarrow P_{i,(j-1)} \geq \pi_{g+1} \geq X \geq P_{ij}$$

$$\longrightarrow K\frac{P_i}{(j-1)} \geq X \geq K\frac{P_i}{(j)} \longrightarrow X\frac{(j-1)}{k} \leq P_i \leq X\frac{j}{k}$$

$\rightarrow$ unser Ziel

Institute of Theoretical Informatics
Algorithmics Group

Partioning Hilfsmethode

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots , \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $\left[\pi_g , \pi_{g+1}\right)$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

- Statt des Tests " $X \geq V$?" benutzen wir den ϵ-Approximierte Test(V).
  $\rightarrow$ x leigt in das Interval $[\pi_g, \pi_{g+1}(1+\epsilon)]$
- Nach Durchführung von Partition(Q,X,$\lceil k(1+\epsilon)\rceil$) :
  $p_i \in R_j$ impliziert dass $X\dfrac{(j-1)}{k} \leq P_i \leq X\dfrac{j}{k(1+\epsilon)}$
  und $p_i \in R_{k(1+\epsilon)+1}$ impliziert $p_i > X$

Partitioning Hilfsmethode - Analyse

Procedure PARTITION (Q,X,K).

0. Let P = $\{ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g, \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Analyse :

Institute of Theoretical Informatics
Algorithmics Group

Partitioning Hilfsmethode - Analyse

Procedure PARTITION (Q,X,K).

0. Let P = $\{\ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g, \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Analyse :

- Schritte 2 hat Komplexität von $\mathcal{O}(\log(mk))$ Tests

Institute of Theoretical Informatics
Algorithmics Group

Partitioning Hilfsmethode - Analyse

Procedure PARTITION (Q,X,K).

0. Let P = $\{ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g, \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Analyse :

- Schritte 2 hat Komplexität von $\mathcal{O}(\log(mk))$ Tests
- Schritte 3 hat Komplexität von $\mathcal{O}(m\log(k))$

Institute of Theoretical Informatics
Algorithmics Group

Partitioning Hilfsmethode - Analyse

Procedure PARTITION (Q,X,K).

0. Let P = $\{ p_{ij} \mid p_{ij} = kp_i/j$, j = 1,...,k, i = 1,...,m $\}$
1. Sort P in increasing order.
   Let $\pi = (\pi_1, \ldots, \pi_l)$ be the resulting sequence of distinct values.
   Extend π by adding $\pi_0 = 0$, $\pi_{l+1} = \infty$.
2. Perform binary search on π to locate the interval $[\pi_g, \pi_{g+1})$ containing X
3. For i = 1,...,m : insert p_i into R_j if $p_{i,(j-1)} > \pi_g \geq p_{ij}$, j = 1, ... ,k.
   Insert p_i into R_{k+1} if $\pi_g < p_{ik} = p_i$

Analyse :

- Schritte 2 hat Komplexität von $\mathcal{O}(\log(mk))$ Tests
- Schritte 3 hat Komplexität von $\mathcal{O}(m\log(k))$
- Insgesamt braucht Partition(Q,X,K) $\mathcal{O}(\log(mk))$ Tests und $\mathcal{O}(m\log(k))$ zusätzliche Operationen

m = $|\mathbb{Q}|$
k = nummer von $\mathbb{R}$ Subsets

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Idee :

$\rightarrow$ Das Algorithmus hat (n-1) Iterationen.

$\rightarrow$ In jeder Iteration i ein Set S_i von einigen (1-i) Pfaden ist berechnet.

$\rightarrow$ Hauptidee ist das Verringern der Größe von den Sets um weniger Komplexität von Algorithmus zu erreichen

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Idee :

$\rightarrow$ Das Algorithmus hat (n-1) Iterationen.

$\rightarrow$ In jeder Iteration i ein Set S_i von einigen (1-i) Pfaden ist berechnet.

$\rightarrow$ Hauptidee ist das Verringern der Größe von den Sets um weniger Komplexität von Algorithmus zu erreichen

Partitioning Algorithm.

0. Let S_1 contain the path consisting of vertex 1.
1. Set $Q_i = \{$ the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of S_j for some $j < i\}$

   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,

   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.

   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.

   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2

   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$

   Set i $\leftarrow$ i + 1 and return to Step 1

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

Wie wird S_i in Schritte 2 gebildet ?

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

Wie wird S_i in Schritte 2 gebildet ?

$\rightarrow$ Entfernen von :

- Pfade von Q_i , die länger als $OPT(1 + \epsilon)$
- Pfade die langsamer als andere sind, die schon in S_i sind, und die Unterschied zwischen ihren längen nicht größer als $\dfrac{OPT\epsilon}{(n-1)(1+\epsilon)}$

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

Wie wird S_i in Schritte 2 gebildet ?

$\rightarrow$ Entfernen von :

- Pfade von Q_i , die länger als $OPT(1 + \epsilon)$
- Pfade die langsamer als andere sind, die schon in S_i sind, und die Unterschied zwischen ihren längen nicht größer als $\frac{OPT\epsilon}{(n-1)(1+\epsilon)}$
- Der akkumulierte relative Fehler $\rightarrow \frac{\epsilon}{1+\epsilon} < \epsilon$

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ - Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ - Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$
- Jede Application von Partition $\rightarrow \mathcal{O}((n^2/\epsilon)\log(n/\epsilon))$ elem. Oper. und $\mathcal{O}(\log(n/\epsilon))$ Tests

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$
- Jede Application von Partition $\rightarrow \mathcal{O}((n^2/\epsilon)\log(n/\epsilon))$ elem. Oper. und $\mathcal{O}(\log(n/\epsilon))$ Tests
- Jeder Test ist $\mathcal{O}(|E|(n/\epsilon)) \quad \rightarrow \quad$ Tests = $\mathcal{O}(|E|(n/\epsilon)\log(n/\epsilon))$

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$
- Jede Application von Partition $\rightarrow \mathcal{O}((n^2/\epsilon)\log(n/\epsilon))$ elem. Oper. und $\mathcal{O}(\log(n/\epsilon))$ Tests
- Jeder Test ist $\mathcal{O}(|E|(n/\epsilon)) \quad \rightarrow \quad$ Tests = $\mathcal{O}(|E|(n/\epsilon)\log(n/\epsilon))$
- Um S_i zu bilden, es nimmt $\mathcal{O}(|Q_i|) = \mathcal{O}(n^2/\epsilon)$

Institute of Theoretical Informatics
Algorithmics Group

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ - Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$
- Jede Application von Partition $\rightarrow \mathcal{O}((n^2/\epsilon)\log(n/\epsilon))$ elem. Oper. und $\mathcal{O}(\log(n/\epsilon))$ Tests
- Jeder Test ist $\mathcal{O}(|E|(n/\epsilon))$ $\rightarrow$ Tests = $\mathcal{O}(|E|(n/\epsilon)\log(n/\epsilon))$
- Um S_i zu bilden, es nimmt $\mathcal{O}(|Q_i|) = \mathcal{O}(n^2/\epsilon)$

  $\rightarrow$ - Insgesamt für jedes i bei Schritte 2 $\rightarrow \mathcal{O}(|E|(n/\epsilon)log(n/\epsilon))$

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

- Größe von S_i ist $\lceil (n-1)(1+\epsilon)/\epsilon \rceil = \mathcal{O}(n/\epsilon)$

  $\rightarrow$ Größe von Q_i ist maximal $\sum_{j<i} |S_j| = \mathcal{O}(n^2/\epsilon)$
- Jede Application von Partition $\rightarrow \mathcal{O}((n^2/\epsilon)\log(n/\epsilon))$ elem. Oper. und $\mathcal{O}(\log(n/\epsilon))$ Tests
- Jeder Test ist $\mathcal{O}(|E|(n/\epsilon)) \rightarrow$ Tests = $\mathcal{O}(|E|(n/\epsilon)\log(n/\epsilon))$
- Um S_i zu bilden, es nimmt $\mathcal{O}(|Q_i|) = \mathcal{O}(n^2/\epsilon)$

  $\rightarrow$ Insgesamt für jedes i bei Schritte 2 $\rightarrow \mathcal{O}(|E|(n/\epsilon)log(n/\epsilon))$

Die Laufzeit vom ganzen Algorithmus $\mathcal{O}(|E|(n^2/\epsilon)log(n/\epsilon))$

Zweite FPAS : Interval Partioning

Partitioning Algorithm.
0. Let S_1 contain the path consisting of vertex 1.
1. Set Q_i = { the lengths of 1-i paths constructed by appending edge (j,i) to a 1-j path of
   S_j for some $j < i$}
   If i = n $\rightarrow$ set $c(P_*) = min\{c(p)|P \in Q_*\}$,
   where $Q_* = \{P \in Q_n | t(P) \leq T\}$, and STOP.
   (If $Q_* = \phi \rightarrow$ then no feasible solution exists.
   Else, P_* is a (1-n) T-path with length $c(p_*) \leq (1 + \epsilon)OPT$)
2. Apply Partition(Q_i, OPT, $\lceil (n-1)(1+\epsilon)/\epsilon \rceil$) using the TEST procedure for the binary search on π in step 2
   Form S_i by picking up a quickest subpath from each R_j, j = 1,...,$\lceil (n-1)(1+\epsilon)/\epsilon \rceil$
   Set i $\leftarrow$ i + 1 and return to Step 1

Vorteil beim zweiten FPAS ?

$\rightarrow$ Die Komplexität vom Rounding Algorithmus (1.FPAS) hängt von der Größe von den Kantenlängen (UB) ab.

$\rightarrow$ Bei Partitioning Algorithmus hängt die Komplexität nur von n und $\frac{1}{\epsilon}$ ab

Zusammenfassung

- Restricted Shortest Path Problem $\rightarrow$ NP-Schwer
- Zwei exakte Pseudopolyomial-Algorithmen
- Erstes FPTAS (Rounding) von Komplexität :

  $\mathcal{O}(\log(\log(B)) \cdot (|E|(n/\varepsilon) + \log(\log(B))$
- Zweites FPTAS (Interval Partitioning) von Komplexität :

  $\mathcal{O}(|E|(n^2/\varepsilon) \cdot \log(n/\varepsilon))$

Institute of Theoretical Informatics
Algorithmics Group

Zusammenfassung

- Restricted Shortest Path Problem $\rightarrow$ NP-Schwer
- Zwei exakte Pseudopolyomial-Algorithmen
- Erstes FPTAS (Rounding) von Komplexität :

  $\mathcal{O}(\log(\log(B)) \cdot (|E|(n/\varepsilon) + \log(\log(B))$
- Zweites FPTAS (Interval Partitioning) von Komplexität :

  $\mathcal{O}(|E|(n^2/\varepsilon) \cdot \log(n/\varepsilon))$

Danke!

Institute of Theoretical Informatics
Algorithmics Group