Approximate Distance Oracles

Michał Zwonek

03 grudnia 2020

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?
- Algorytm Floyd-Warshall'a, obliczamy w $O(n^3)$ macierz $O(n^2)$ wszystkich odległości.

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?
- Algorytm Floyd-Warshall'a, obliczamy w $O(n^3)$ macierz $O(n^2)$ wszystkich odległości.
- Problem jak n jest duże! $O(n^2)$ dla $n = 10^6$ to $4TB$. Dużo.

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?
- Algorytm Floyd-Warshall'a, obliczamy w $O(n^3)$ macierz $O(n^2)$ wszystkich odległości.
- Problem jak n jest duże! $O(n^2)$ dla $n = 10^6$ to $4TB$. Dużo.
- Trzymanie w macierzy $\rightarrow$ degradacja czasu zapytania, nadal $O(1)$, ale w praktyce coraz gorzej (trudno $4TB$ mieć w RAM).

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?
- Algorytm Floyd-Warshall'a, obliczamy w $O(n^3)$ macierz $O(n^2)$ wszystkich odległości.
- Problem jak n jest duże! $O(n^2)$ dla $n = 10^6$ to $4TB$. Dużo.
- Trzymanie w macierzy $\rightarrow$ degradacja czasu zapytania, nadal $O(1)$, ale w praktyce coraz gorzej (trudno $4TB$ mieć w RAM).
- Drugi problem? Jak trzymać ścieżkę?

Problem wyroczni

- Mamy graf (lub ogólnie metrykę), dla którego chcemy wiedzieć ile wynosi $\delta(u, v)$.
- Poza tym, chcemy wiedzieć jak wygląda taka trasa!
- Co możemy zrobić?
- Algorytm Floyd-Warshall'a, obliczamy w $O(n^3)$ macierz $O(n^2)$ wszystkich odległości.
- Problem jak n jest duże! $O(n^2)$ dla $n = 10^6$ to $4TB$. Dużo.
- Trzymanie w macierzy $\rightarrow$ degradacja czasu zapytania, nadal $O(1)$, ale w praktyce coraz gorzej (trudno $4TB$ mieć w RAM).
- Drugi problem? Jak trzymać ścieżkę?
- Alternatywa?

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?
- Użyjmy Algorytmu Dijkistry!

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?
- Użyjmy Algorytmu Dijkistry!
- Preprocessing $O(1)$, świetnie!

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?
- Użyjmy Algorytmu Dijkistry!
- Preprocessing $O(1)$, świetnie!
- Pamięć? $O(n+m)$, optymalnie!

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?
- Użyjmy Algorytmu Dijkistry!
- Preprocessing $O(1)$, świetnie!
- Pamięć? $O(n+m)$, optymalnie!
- Czas zapytania? :(

Użycie algorytmów najkrótszych ścieżek

- Floyd-Warshall $O(n^3)$ preprocessing, $O(n^2)$ pamięć, $O(1)$ zapytanie.
- Alternatywa?
- Użyjmy Algorytmu Dijkistry!
- Preprocessing $O(1)$, świetnie!
- Pamięć? $O(n+m)$, optymalnie!
- Czas zapytania? :(
- $O(m + nlogn)$, trochę słabo...

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:
  1. Preprocessing, przydałoby się lepiej niż $O(n^3)$...

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:
  1. Preprocessing, przydałoby się lepiej niż $O(n^3)$...
  2. Pamięć, przydałoby się lepiej niż $O(n^2)$...

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:
  1. Preprocessing, przydałoby się lepiej niż $O(n^3)$...
  2. Pamięć, przydałoby się lepiej niż $O(n^2)$...
  3. Czas zapytania, przydałoby się lepiej niż $O(n+m)$...

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:
  1. Preprocessing, przydałoby się lepiej niż $O(n^3)$...
  2. Pamięć, przydałoby się lepiej niż $O(n^2)$...
  3. Czas zapytania, przydałoby się lepiej niż $O(n+m)$...
- Możemy się pokusić, by uzyskać dobre wyniki, na nieco gorsze odpowiedzi niż faktyczne.

Pomysł?

- Spróbujmy wstrzelić się pomiędzy te dwa podejścia!
- Będziemy patrzeć na:
  1. Preprocessing, przydałoby się lepiej niż $O(n^3)$...
  2. Pamięć, przydałoby się lepiej niż $O(n^2)$...
  3. Czas zapytania, przydałoby się lepiej niż $O(n+m)$...
- Możemy się pokusić, by uzyskać dobre wyniki, na nieco gorsze odpowiedzi niż faktyczne.
- Zgodzimy się, żeby algorytm zwracał nam wyniki gorsze niż optymalny (niewiele gorsze), ale wraz z dowodem poprawności ich długości.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.
- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.
- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.
- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Złożonośc pamięciowa $O(kn^{1+1/k})$.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.
- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.

Bohater referatu - Approximate Distance Oracle

- Approximate Distance Oracle (January 2005, Mikkel Thorup, Uri Zwick)
- Wyrocznia odpowiadająca na zapytania o odległośc $\delta(u, v)$ i zwracająca $\overline{\delta}(u, v)$.
- Wynik, może być nieoptymalny, ale będzie zgodne ze *stretch*'em ω, takim, że dla każdego u, v:

$$\delta(u, v) \leq \overline{\delta}(u, v) \leq \omega \cdot \delta(u, v).$$

- *Stretch* dla wyroczni będzie parametryzowany liczba k.
- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$, czyli stałe?

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.
- Dla $k = 1$ mamy $O(n^{1+1/1}) = O(n^2)$ pamięć, $O(mn)$ preprocesing i $O(1)$ zapytanie, co to?

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.
- Dla $k = 1$ mamy $O(n^{1+1/1}) = O(n^2)$ pamięć, $O(mn)$ preprocesing i $O(1)$ zapytanie, co to?
- Równoważny algorytm dla Floyd-Warshall'a.

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.
- Dla $k = 1$ mamy $O(n^{1+1/1}) = O(n^2)$ pamięć, $O(mn)$ preprocesing i $O(1)$ zapytanie, co to?
- Równoważny algorytm dla Floyd-Warshall'a.
- Dla $k = 2$ mamy $O(n\sqrt{n})$ pamięć, ale stretch wynosi $\omega = 3$.

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.
- Dla $k = 1$ mamy $O(n^{1+1/1}) = O(n^2)$ pamięć, $O(mn)$ preprocesing i $O(1)$ zapytanie, co to?
- Równoważny algorytm dla Floyd-Warshall'a.
- Dla $k = 2$ mamy $O(n\sqrt{n})$ pamięć, ale stretch wynosi $\omega = 3$.
- Dla $k = \log n$ mamy $O(\log n \cdot n \cdot n^{1/\log n}) = O(\log n \cdot n \cdot 2) = O(n \log n)$ pamięć, zapytanie $O(\log n)$ oraz stretch $\omega = \log n$.

Różne k

- Dla dowolnego k istnieje wyrocznia aproksymująca odległości spełniająca:
  1. $\omega = (2k - 1)$.
  2. Oczekiwana złożonośc pamięciowa $O(kn^{1+1/k})$.
  3. Preprocessing $O(kmn^{1/k})$.
  4. Zapytanie $O(k)$.
- Dla $k = 1$ mamy $O(n^{1+1/1}) = O(n^2)$ pamięć, $O(mn)$ preprocesing i $O(1)$ zapytanie, co to?
- Równoważny algorytm dla Floyd-Warshall'a.
- Dla $k = 2$ mamy $O(n\sqrt{n})$ pamięć, ale stretch wynosi $\omega = 3$.
- Dla $k = \log n$ mamy $O(\log n \cdot n \cdot n^{1/\log n}) = O(\log n \cdot n \cdot 2) = O(n \log n)$ pamięć, zapytanie $O(\log n)$ oraz stretch $\omega = \log n$.
- $k = \omega(\log n)$?

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...
- Najpierw, skonstruujmy zbiory $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...
- Najpierw, skonstruujmy zbiory $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...
- Najpierw, skonstruujmy zbiory $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Czyli, część wierzchołków z A_i przetrwa do A_{i+1}, każdy ma tę samą szansę $p = n^{-1/k}$.

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...
- Najpierw, skonstruujmy zbiory $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Czyli, część wierzchołków z A_i przetrwa do A_{i+1}, każdy ma tę samą szansę $p = n^{-1/k}$.
- Wtedy: $\mathbb{E}(\#A_i) = n \cdot p^i = n \cdot n^{-i/k} = n^{1-i/k}$.

Główny pomysł

- Randomizowany algorytm (można go zderandomizować małym kosztem czasu konstrukcji).
- *Bunches*...
- Najpierw, skonstruujmy zbiory $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Czyli, część wierzchołków z A_i przetrwa do A_{i+1}, każdy ma tę samą szansę $p = n^{-1/k}$.
- Wtedy: $\mathbb{E}(\#A_i) = n \cdot p^i = n \cdot n^{-i/k} = n^{1-i/k}$.
- Intuicja? Zbiory A_i maleją w "regularny" sposób, aż do zbioru pustego.

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

- Gdzie $B_i(v) = \{w \in A_i \setminus A_{i+1} | \delta(w, v) < \delta(A_{i+1}, v)\}$.

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

- Gdzie $B_i(v) = \{w \in A_i \setminus A_{i+1} | \delta(w, v) < \delta(A_{i+1}, v)\}$.
- W $B_i(v)$ są te wierzchołki w, które znikną w następnej warstwie, a są bliżej niż najbliższy z następnej warstwy.

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

- Gdzie $B_i(v) = \{w \in A_i \setminus A_{i+1} | \delta(w, v) < \delta(A_{i+1}, v)\}$.
- W $B_i(v)$ są te wierzchołki w, które znikną w następnej warstwie, a są bliżej niż najbliższy z następnej warstwy.
- Łatwo pokazać, że $\mathbb{E}[\#B_i(v)] \leq n^{1/k}$, więc $\mathbb{E}[\#B(v)] \leq kn^{1/k}$.

Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

- Gdzie $B_i(v) = \{w \in A_i \setminus A_{i+1} | \delta(w, v) < \delta(A_{i+1}, v)\}$.
- W $B_i(v)$ są te wierzchołki w, które znikną w następnej warstwie, a są bliżej niż najbliższy z następnej warstwy.
- Łatwo pokazać, że $\mathbb{E}[\#B_i(v)] \leq n^{1/k}$, więc $\mathbb{E}[\#B(v)] \leq kn^{1/k}$.
- Jeśli dla każdego wierzchołka będziemy trzymać $B(v)$ zajmie nam to $O(kn^{1+1/k})$ pamięci.

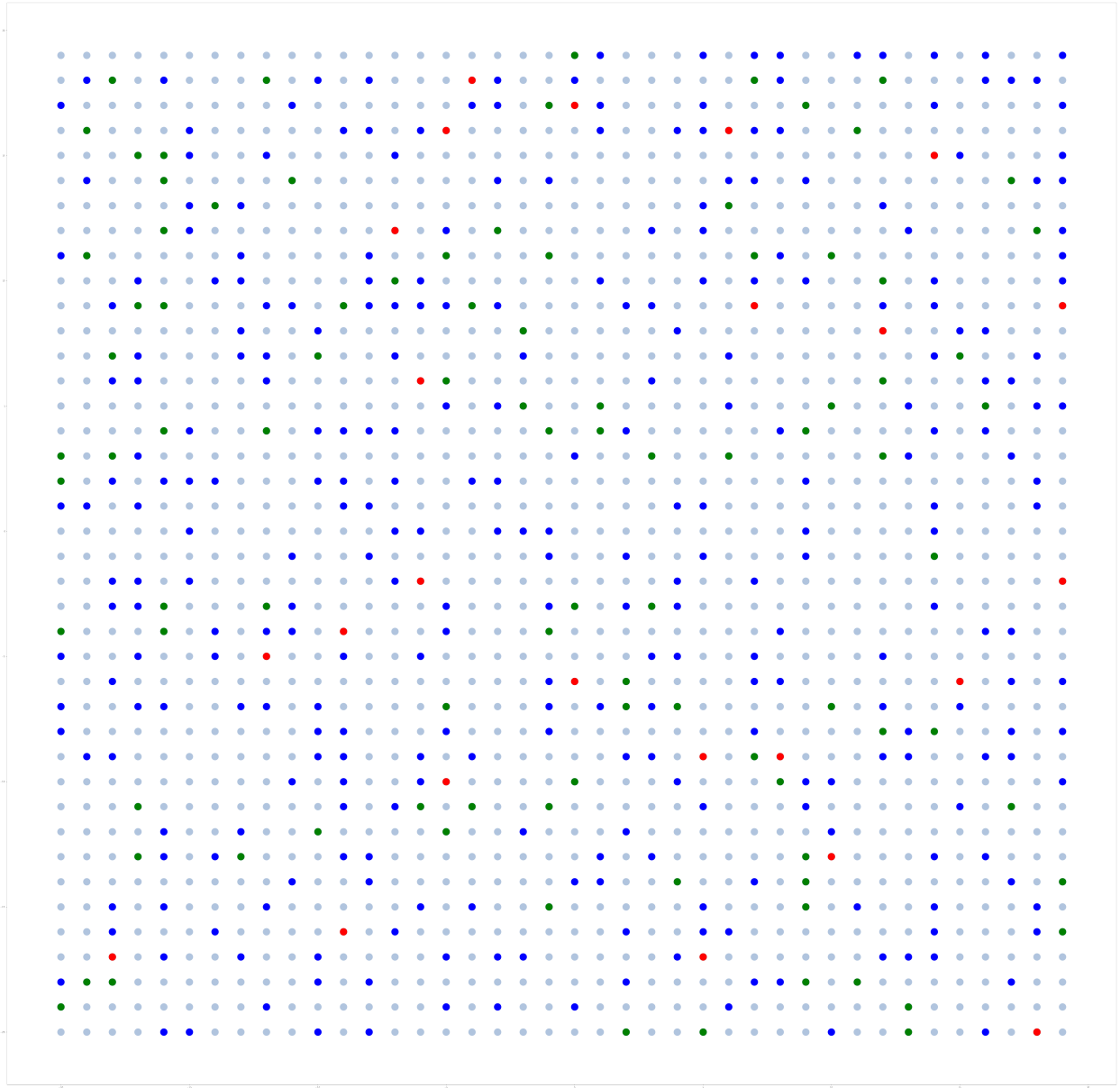
Bunches

- $V = A_0 \supset ... \supset A_{n-1} \supset A_n = \emptyset$.
- $\forall_{0<i<k} \forall v : P(v \in A_i | v \in A_{i-1}) = n^{-1/k}$
- Zdefiniujemy *Bunch* dla wierzchołka jako

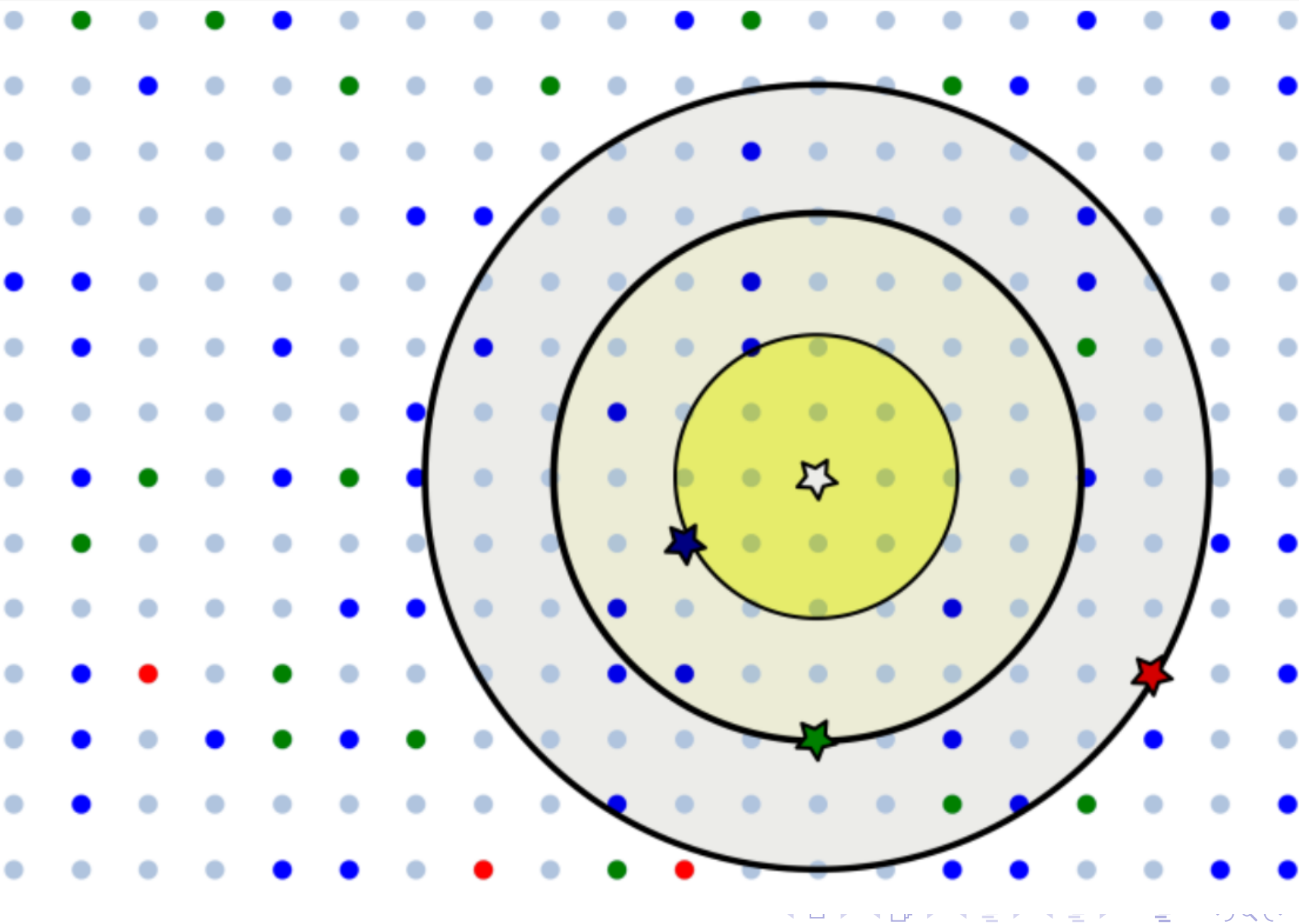
$$B(v) = \cup_{i=0}^{k-1} B_i(v).$$

- Gdzie $B_i(v) = \{w \in A_i \setminus A_{i+1} | \delta(w, v) < \delta(A_{i+1}, v)\}$.
- W $B_i(v)$ są te wierzchołki w, które znikną w następnej warstwie, a są bliżej niż najbliższy z następnej warstwy.
- Łatwo pokazać, że $\mathbb{E}[\#B_i(v)] \leq n^{1/k}$, więc $\mathbb{E}[\#B(v)] \leq kn^{1/k}$.
- Jeśli dla każdego wierzchołka będziemy trzymać $B(v)$ zajmie nam to $O(kn^{1+1/k})$ pamięci.
- Dodatkowo potrzebujemy dla każdego i, v znać świadka $p_i(v)$ takiego, że $\delta(A_i, v) = \delta(p_i(v), v)$.

Bunches, diagram



Bunches, diagram



Query

- Algorytm $query(u, v)$ działający w $O(k)$.

Query

- Algorytm $query(u, v)$ działający w $O(k)$.
- **procedure** query$_k(u, v)$
  $i \leftarrow 0$
  $w \leftarrow u$
  while $w \notin B(v)$ **do**
    $i \leftarrow i + 1$
    swap(u, v)
    $w \leftarrow p_i(u)$
  end while
  return $\delta(u, w) + \delta(w, v)$
end procedure

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.
- Czyli
  $\overline{\delta}(u, v) = \delta(u, w) + \delta(w, v) \leq \delta(u, w) + (\delta(w, u) + \delta(w, v)) =$
  $2\delta(u, w) + \delta(u, v) \leq (2k-1)\delta(u, v)$.

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.
- Czyli
  $\overline{\delta}(u, v) = \delta(u, w) + \delta(w, v) \leq \delta(u, w) + (\delta(w, u) + \delta(w, v)) =$
  $2\delta(u, w) + \delta(u, v) \leq (2k-1)\delta(u, v)$.
- Co daje nam stretch $\omega = 2k - 1$.

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.
- Czyli
  $\overline{\delta}(u, v) = \delta(u, w) + \delta(w, v) \leq \delta(u, w) + (\delta(w, u) + \delta(w, v)) =$
  $2\delta(u, w) + \delta(u, v) \leq (2k-1)\delta(u, v)$.
- Co daje nam stretch $\omega = 2k - 1$.
- Tak naprawdę dokładna konstrukcja bunchy nie jest istotna dla stretcha. Co to znaczy?

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.
- Czyli $\overline{\delta}(u, v) = \delta(u, w) + \delta(w, v) \leq \delta(u, w) + (\delta(w, u) + \delta(w, v)) = 2\delta(u, w) + \delta(u, v) \leq (2k-1)\delta(u, v)$.
- Co daje nam stretch $\omega = 2k - 1$.
- Tak naprawdę dokładna konstrukcja bunchy nie jest istotna dla stretcha. Co to znaczy?
- Można też 'poprawić' jakość rozwiązań.

Stretch dla query

- W każdej iteracji pętli while dystans między u a w zwiększy się o co najwyżej $\delta(u, v)$.
- Wtedy pod koniec mamy, że $\delta(u, w) \leq (k-1)\delta(u, v)$.
- Czyli $\overline{\delta}(u, v) = \delta(u, w) + \delta(w, v) \leq \delta(u, w) + (\delta(w, u) + \delta(w, v)) = 2\delta(u, w) + \delta(u, v) \leq (2k-1)\delta(u, v)$.
- Co daje nam stretch $\omega = 2k - 1$.
- Tak naprawdę dokładna konstrukcja bunchy nie jest istotna dla stretcha. Co to znaczy?
- Można też 'poprawić' jakość rozwiązań.
- Mimo, że $\delta(u, v) = \delta(v, u)$ to może być, że $\overline{\delta}(u, v) \neq \overline{\delta}(v, u)$.

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.
- Łatwo obliczyć $C(w)$, po prostu SSSP?

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.
- Łatwo obliczyć $C(w)$, po prostu SSSP?
- Mając wszystkie $C(w)$ możemy łatwo skonstruować wszystkie $B(v)$, tak też wygląda preprocessing!

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.
- Łatwo obliczyć $C(w)$, po prostu SSSP?
- Mając wszystkie $C(w)$ możemy łatwo skonstruować wszystkie $B(v)$, tak też wygląda preprocessing!
- Zapamiętujemy też drzewo najkrótszej ścieżki rozpinające $C(w)$ dla każdego w.

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.
- Łatwo obliczyć $C(w)$, po prostu SSSP?
- Mając wszystkie $C(w)$ możemy łatwo skonstruować wszystkie $B(v)$, tak też wygląda preprocessing!
- Zapamiętujemy też drzewo najkrótszej ścieżki rozpinające $C(w)$ dla każdego w.
- Dokładnie z tego drzewa korzystamy konstruując ścieżkę między u, v.

Konstrukcja ścieżki

- Konstrukcja ścieżki jest trochę bardziej techniczna, opiera się na konstrukcji zbiorów Cluster.

$$C(w \in A_i \setminus A_{i+1}) = \{v \in V \mid \delta(w, v) < \delta(A_{i+1}, v)\}$$

- Co ciekawe są one dualne do Bunches: Cluster $C(w)$ spełnia, że $v \in C(w) \leftrightarrow w \in B(v)$.
- $w \in B(v) \Leftrightarrow w \in A_i \setminus A_{i+1} \wedge \delta(w, v) < \delta(A_{i+1}, v) \Leftrightarrow v \in C(w)$.
- Łatwo obliczyć $C(w)$, po prostu SSSP?
- Mając wszystkie $C(w)$ możemy łatwo skonstruować wszystkie $B(v)$, tak też wygląda preprocessing!
- Zapamiętujemy też drzewo najkrótszej ścieżki rozpinające $C(w)$ dla każdego w.
- Dokładnie z tego drzewa korzystamy konstruując ścieżkę między u, v.
- Ciekawym jest, że trasa skonstruowana może być lepsza niż $\overline{\delta}(u, v)$, ale nie gorsza!

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)
- Wyrafinowany i nieoczywisty Binary Search.

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)
- Wyrafinowany i nieoczywisty Binary Search.
- Query time $O(1)$, która jest hybrydą oryginalnej wyroczni i wyroczni o czasie $O(1)$ z $128k$ stretch. (Shiri Chechik 2014)

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)
- Wyrafinowany i nieoczywisty Binary Search.
- Query time $O(1)$, która jest hybrydą oryginalnej wyroczni i wyroczni o czasie $O(1)$ z $128k$ stretch. (Shiri Chechik 2014)
- Niesamowicie skomplikowany i nieklepalny (przynajmniej dla mnie).

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)
- Wyrafinowany i nieoczywisty Binary Search.
- Query time $O(1)$, która jest hybrydą oryginalnej wyroczni i wyroczni o czasie $O(1)$ z $128k$ stretch. (Shiri Chechik 2014)
- Niesamowicie skomplikowany i nieklepalny (przynajmniej dla mnie).
- Kiedy przydałby się $O(\log k)$? Żeby miał sens to k powinno być znacząco większe niż 10.

Ulepszenia Oryginalnego Oracle'a

- Query time $O(\log k)$ (Christian Wulff-Nilsen 2012)
- Wyrafinowany i nieoczywisty Binary Search.
- Query time $O(1)$, która jest hybrydą oryginalnej wyroczni i wyroczni o czasie $O(1)$ z $128k$ stretch. (Shiri Chechik 2014)
- Niesamowicie skomplikowany i nieklepalny (przynajmniej dla mnie).
- Kiedy przydałby się $O(\log k)$? Żeby miał sens to k powinno być znacząco większe niż 10.
- Ale dla $k = 10$ mamy złożoność $O(kn^{1.10})$, wiele więcej nie wyciągniemy...

Implementacja

- Wyrocznie $O(n)$, $O(\log n)$ i ich polepszane warianty.

Implementacja

- Wyrocznie $O(n)$, $O(\log n)$ i ich polepszane warianty.
- Konstruowanie ścieżki.

Implementacja

- Wyrocznie $O(n)$, $O(\log n)$ i ich polepszane warianty.
- Konstruowanie ścieżki.
- Wszystko w C++ z własnymi alokatorami pamięci - weryfikacja złożoności pamięciowej...

Implementacja

- Wyrocznie $O(n)$, $O(\log n)$ i ich polepszane warianty.
- Konstruowanie ścieżki.
- Wszystko w C++ z własnymi alokatorami pamięci - weryfikacja złożoności pamięciowej...
- Testowane na różnych grafach, losowych, ale też na grafie sieci dróg USA.

Wyniki

- $\Phi(u,v) = \overline{\delta}(u,v)/\delta(u,v)$

$$Err(DO) = \mathbb{E}[(\Phi(u,v)-1)^2] = \left(\sum_{u,v}^{V}(\Phi(u,v)-1)^2\right)\cdot(\#V)^{-2}$$

Wyniki

- $\Phi(u,v) = \overline{\delta}(u,v)/\delta(u,v)$

$$Err(DO) = \mathbb{E}[(\Phi(u,v)-1)^2] = \left(\sum_{u,v}^{V}(\Phi(u,v)-1)^2\right)\cdot(\#V)^{-2}$$

- Faktyczne wyniki dużo lepsze niż pesymistyczny stretch.

Wyniki

- $\Phi(u,v) = \overline{\delta}(u,v)/\delta(u,v)$

$$Err(DO) = \mathbb{E}[(\Phi(u,v)-1)^2] = \left(\sum_{u,v}^{V}(\Phi(u,v)-1)^2\right)\cdot(\#V)^{-2}$$

- Faktyczne wyniki dużo lepsze niż pesymistyczny stretch.
- Średnia *Err* dla niepolepszanej wyroczni wyniosła 0.168579.

Wyniki

- $\Phi(u,v) = \overline{\delta}(u,v)/\delta(u,v)$

$$Err(DO) = \mathbb{E}[(\Phi(u,v)-1)^2] = \left(\sum_{u,v}^{V}(\Phi(u,v)-1)^2\right)\cdot(\#V)^{-2}$$

- Faktyczne wyniki dużo lepsze niż pesymistyczny stretch.
- Średnia *Err* dla niepolepszanej wyroczni wyniosła 0.168579.
- Średnia *Err* dla polepszanych wyroczni wyniosła około 0.100316.

Wyniki

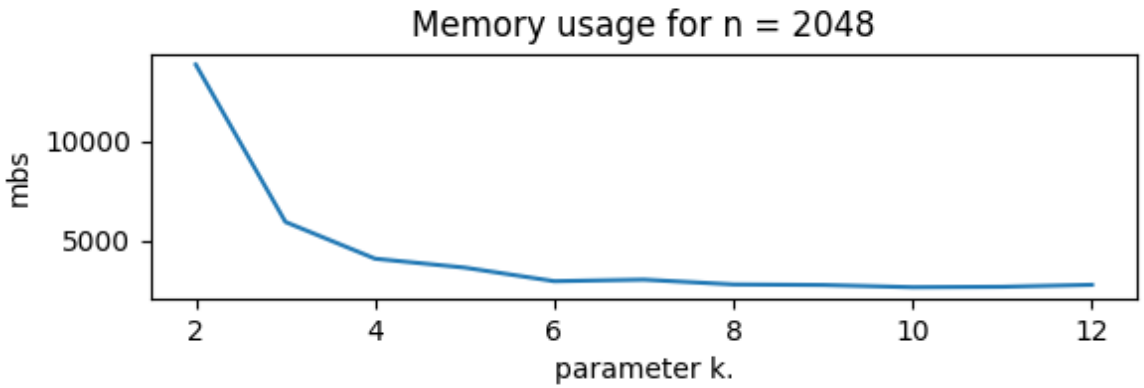
- $\Phi(u,v) = \overline{\delta}(u,v)/\delta(u,v)$

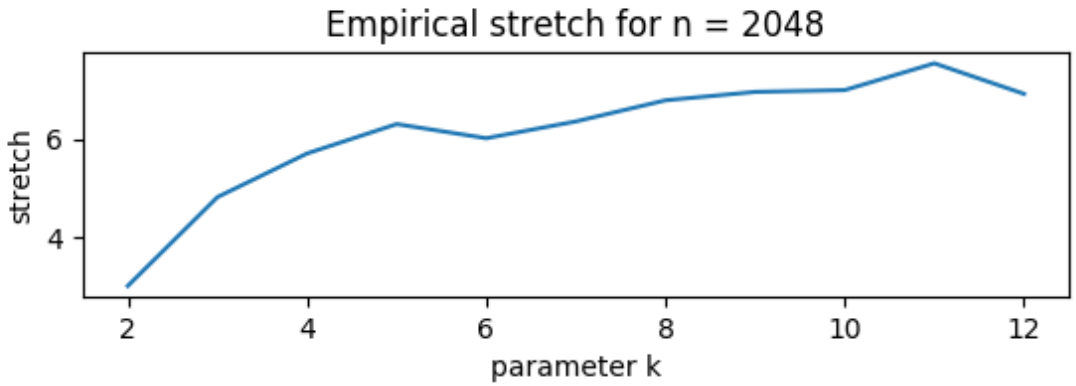
$$Err(DO) = \mathbb{E}[(\Phi(u,v)-1)^2] = \left(\sum_{u,v}^{V}(\Phi(u,v)-1)^2\right)\cdot(\#V)^{-2}$$

- Faktyczne wyniki dużo lepsze niż pesymistyczny stretch.
- Średnia *Err* dla niepolepszanej wyroczni wyniosła 0.168579.
- Średnia *Err* dla polepszanych wyroczni wyniosła około 0.100316.
- $k \in \{1,2,3\}$ wydaje się być wystarczająco dobre dla większości praktycznych zastosowań.

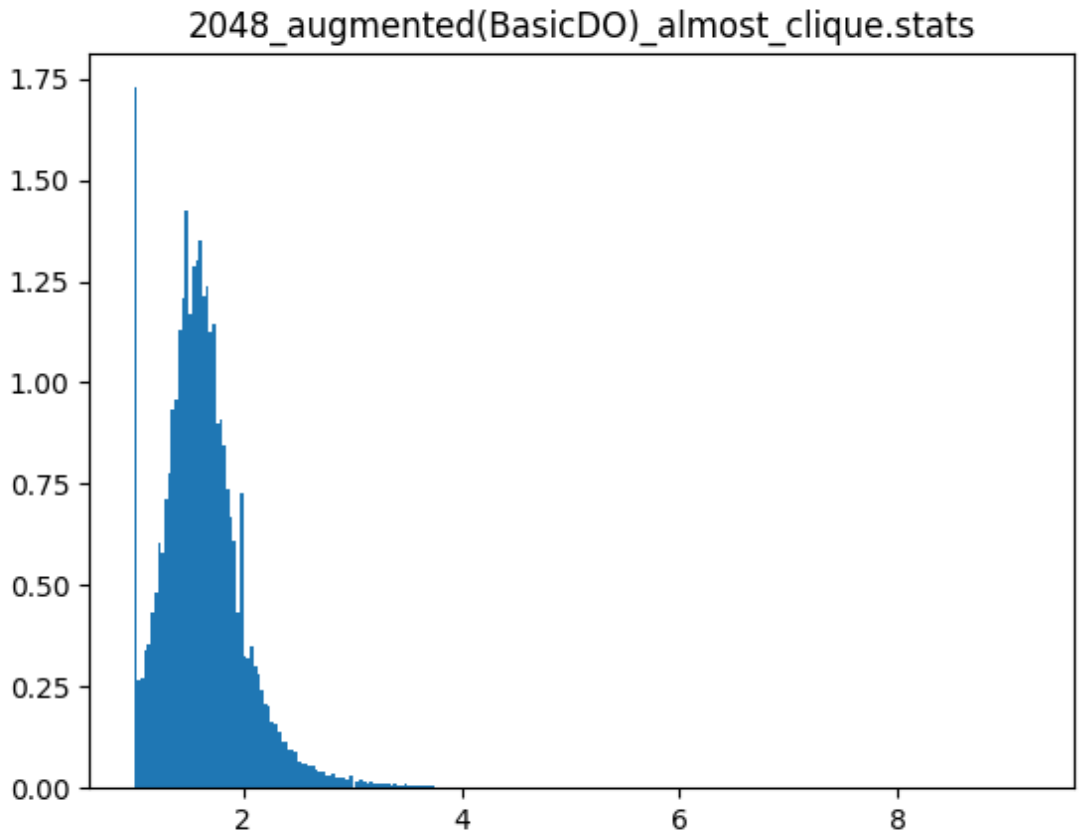
Stretch, a pamięć

- Empirical stretch to najgorszy uzyskany stretch $\Phi(u, v)$.

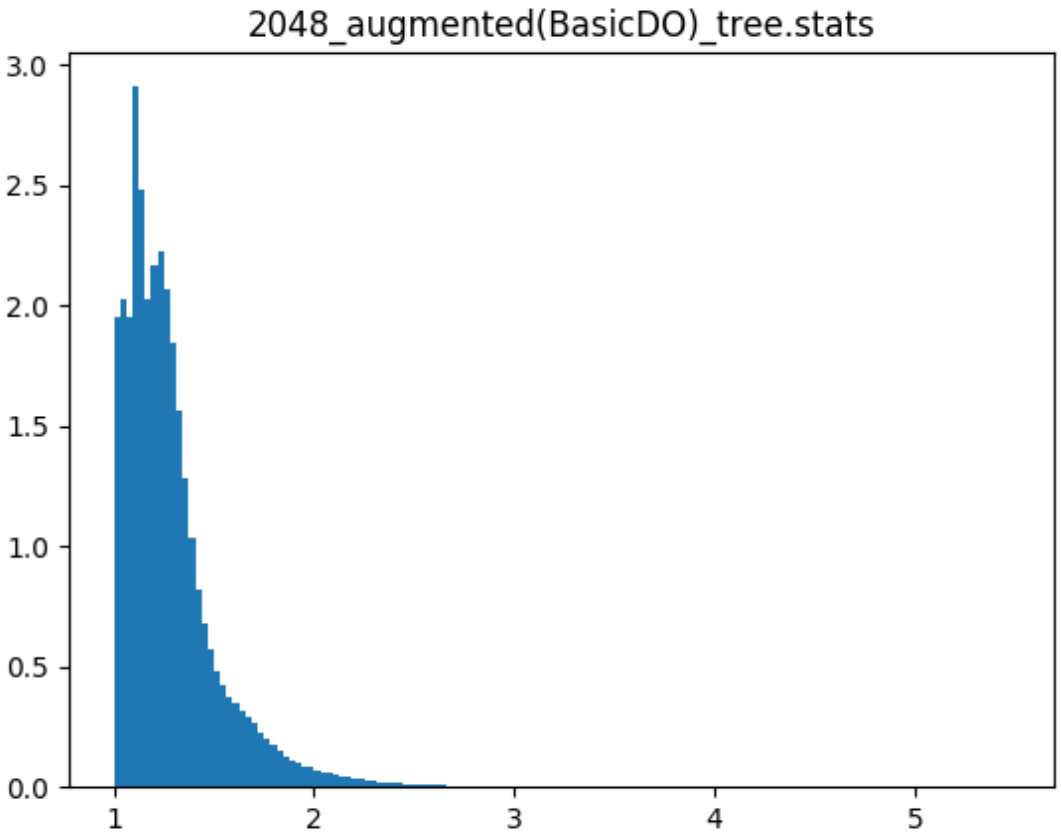




Rozkład $\Phi(u, v)$



Rozkład $\Phi(u, v)$

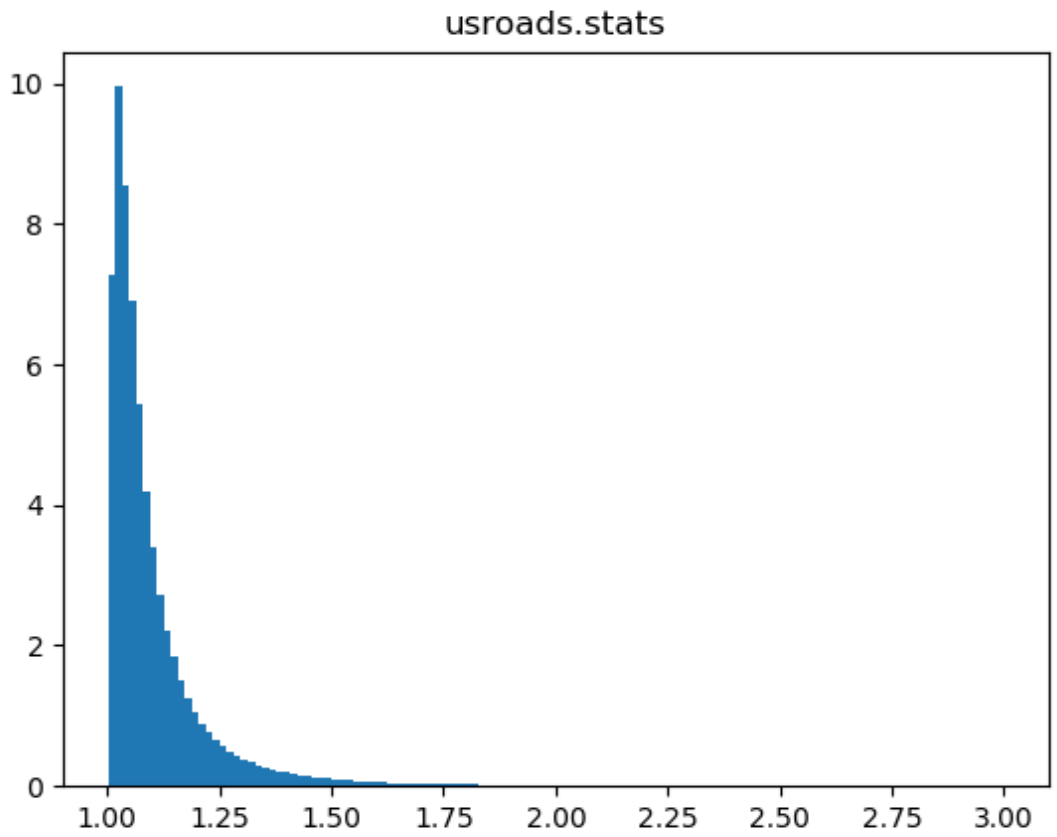


Rozkład $\Phi(u, v)$ dla USRoads

- $n = 129164$, $m = 165435$, $k = 3$, faktycznie najgorszy osiągniety stretch to 4.38462

Rozkład $\Phi(u, v)$ dla USRoads

- $n = 129164$, $m = 165435$, $k = 3$, faktycznie najgorszy osiągniety stretch to 4.38462



Dziękuję za uwagę

- Dziękuję za uwagę!