

Backward analysis

Lecture 13

October 8, 2015

13.1: Some more probability

Some more probability

Lemma

$\mathcal{E}_1, \dots, \mathcal{E}_n$: n events (not necessarily independent). Then,

$$\Pr\left[\bigcap_{i=1}^n \mathcal{E}_i\right] = \Pr\left[\mathcal{E}_1\right] * \Pr\left[\mathcal{E}_2 \mid \mathcal{E}_1\right] * \Pr\left[\mathcal{E}_3 \mid \mathcal{E}_1 \cap \mathcal{E}_2\right] * \dots \\ * \Pr\left[\mathcal{E}_n \mid \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{n-1}\right].$$

13.2: Backward analysis

Backward analysis

- ① $\mathbf{P} = \langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be a random ordering of n distinct numbers.
- ② $X_i = 1 \iff \mathbf{p}_i$ is smaller than $\mathbf{p}_1, \dots, \mathbf{p}_{i-1}$.

③ Lemma

$$\Pr[X_i = 1] = 1/i.$$

Backward analysis

- ① $\mathbf{P} = \langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be a random ordering of n distinct numbers.
- ② $X_i = 1 \iff \mathbf{p}_i$ is smaller than $\mathbf{p}_1, \dots, \mathbf{p}_{i-1}$.

③ Lemma

$$\Pr[X_i = 1] = 1/i.$$

Backward analysis

- ① $\mathbf{P} = \langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be a random ordering of n distinct numbers.
- ② $X_i = 1 \iff \mathbf{p}_i$ is smaller than $\mathbf{p}_1, \dots, \mathbf{p}_{i-1}$.

③ Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof...

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[p_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i$.

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i.$

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i.$

$$\Pr[\mathbf{p}_i = \min(\mathbf{P}_i)]$$

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i$.

$$\begin{aligned} & \Pr[\mathbf{p}_i = \min(\mathbf{P}_i)] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i) = S] \Pr[S] \end{aligned}$$

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i$.

$$\begin{aligned} & \Pr[\mathbf{p}_i = \min(\mathbf{P}_i)] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i) = S] \Pr[S] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \frac{1}{i} \Pr[S] \end{aligned}$$

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i$.

$$\begin{aligned} & \Pr[\mathbf{p}_i = \min(\mathbf{P}_i)] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i) = S] \Pr[S] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \frac{1}{i} \Pr[S] = \frac{1}{i}. \end{aligned}$$

Lemma

$$\Pr[X_i = 1] = 1/i.$$

Proof.

- 1 Fix elements appearing in $\text{set}(\mathbf{P}_i) = \{s_1, \dots, s_i\}$.
- 2 $\Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i)] = 1/i$.

$$\begin{aligned} & \Pr[\mathbf{p}_i = \min(\mathbf{P}_i)] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \Pr[\mathbf{p}_i = \min(\mathbf{P}_i) \mid \text{set}(\mathbf{P}_i) = S] \Pr[S] \\ &= \sum_{S \subseteq \mathbf{P}, |S|=i} \frac{1}{i} \Pr[S] = \frac{1}{i}. \quad \square \end{aligned}$$

of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $E[Y] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n 1/i \leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $E[Y] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n 1/i \leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

① $Y = \sum_{i=1}^n X_i.$

② $E[Y] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n 1/i$
 $\leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $E[Y] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n 1/i \leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $\mathbf{E}[Y] = \mathbf{E}[\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbf{E}[X_i] = \sum_{i=1}^n 1/i$
 $\leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $\mathbf{E}[Y] = \mathbf{E}[\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbf{E}[X_i] = \sum_{i=1}^n 1/i$
 $\leq \ln n + 1.$



of times...

...the minimum changes in a random permutation...

Theorem

In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.

Proof.

- 1 $Y = \sum_{i=1}^n X_i.$
- 2 $\mathbf{E}[Y] = \mathbf{E}[\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbf{E}[X_i] = \sum_{i=1}^n 1/i \leq \ln n + 1.$



Lemma

$\Pi = \pi_1 \dots \pi_n$: random permutation of $\{1, \dots, n\}$. X_i : indicator variable if π_i is the smallest number in $\{\pi_1, \dots, \pi_i\}$, for $\forall i$.

Then $Z = \sum_{i=1}^n X_i = O(\log n)$., w.h.p. (i.e., $\geq 1 - 1/n^{O(1)}$).

proof

- 1 \mathcal{E}_i : the event that $X_i = 1$, for $i = 1, \dots, n$.
- 2 Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
- 3 Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n .
- 4 Next, pick a random permutation of the remaining numbers and set the first number as π_{n-1} in output permutation.
- 5 Repeat this process till we generate the whole permutation.

Lemma

$\Pi = \pi_1 \dots \pi_n$: random permutation of $\{1, \dots, n\}$. X_i : indicator variable if π_i is the smallest number in $\{\pi_1, \dots, \pi_i\}$, for $\forall i$.

Then $Z = \sum_{i=1}^n X_i = O(\log n)$., w.h.p. (i.e., $\geq 1 - 1/n^{O(1)}$).

proof

- 1 \mathcal{E}_i : the event that $X_i = 1$, for $i = 1, \dots, n$.
- 2 Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
- 3 Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n .
- 4 Next, pick a random permutation of the remaining numbers and set the first number as π_{n-1} in output permutation.
- 5 Repeat this process till we generate the whole permutation.

Lemma

$\Pi = \pi_1 \dots \pi_n$: random permutation of $\{1, \dots, n\}$. X_i : indicator variable if π_i is the smallest number in $\{\pi_1, \dots, \pi_i\}$, for $\forall i$.

Then $Z = \sum_{i=1}^n X_i = O(\log n)$., w.h.p. (i.e., $\geq 1 - 1/n^{O(1)}$).

proof

- 1 \mathcal{E}_i : the event that $X_i = 1$, for $i = 1, \dots, n$.
- 2 Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
- 3 Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n .
- 4 Next, pick a random permutation of the remaining numbers and set the first number as π_{n-1} in output permutation.
- 5 Repeat this process till we generate the whole permutation.

Lemma

$\Pi = \pi_1 \dots \pi_n$: random permutation of $\{1, \dots, n\}$. X_i : indicator variable if π_i is the smallest number in $\{\pi_1, \dots, \pi_i\}$, for $\forall i$.

Then $Z = \sum_{i=1}^n X_i = O(\log n)$, w.h.p. (i.e., $\geq 1 - 1/n^{O(1)}$).

proof

- 1 \mathcal{E}_i : the event that $X_i = 1$, for $i = 1, \dots, n$.
- 2 Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
- 3 Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n .
- 4 Next, pick a random permutation of the remaining numbers and set the first number as π_{n-1} in output permutation.
- 5 Repeat this process till we generate the whole permutation.

Lemma

$\Pi = \pi_1 \dots \pi_n$: random permutation of $\{1, \dots, n\}$. X_i : indicator variable if π_i is the smallest number in $\{\pi_1, \dots, \pi_i\}$, for $\forall i$.

Then $Z = \sum_{i=1}^n X_i = O(\log n)$., w.h.p. (i.e., $\geq 1 - 1/n^{O(1)}$).

proof

- 1 \mathcal{E}_i : the event that $X_i = 1$, for $i = 1, \dots, n$.
- 2 Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
- 3 Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n .
- 4 Next, pick a random permutation of the remaining numbers and set the first number as π_{n-1} in output permutation.
- 5 Repeat this process till we generate the whole permutation.

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

- ① For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- ② ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- ③ By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- ④ \implies variables X_1, \dots, X_n are independent.
- ⑤ Result readily follows from Chernoff's inequality. ■

Proof continued...

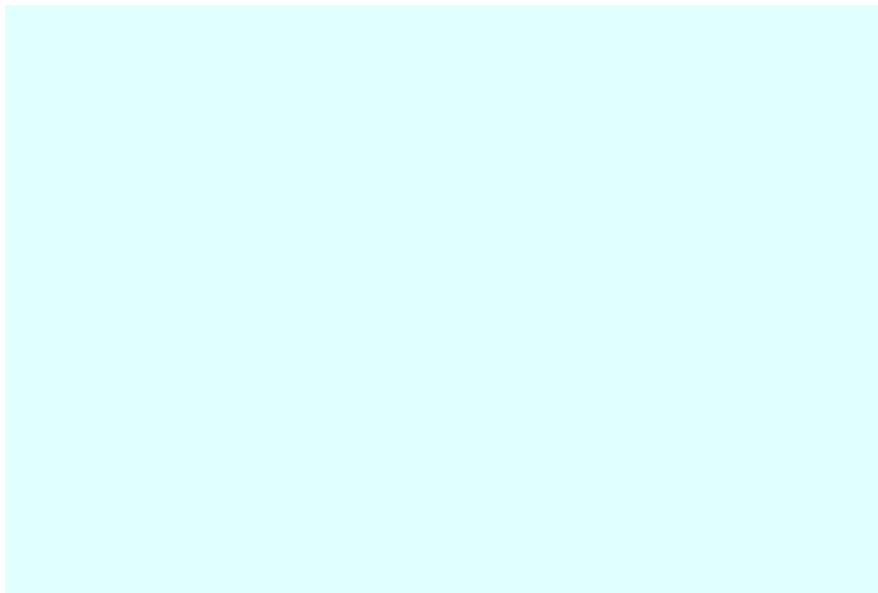
- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

Proof continued...

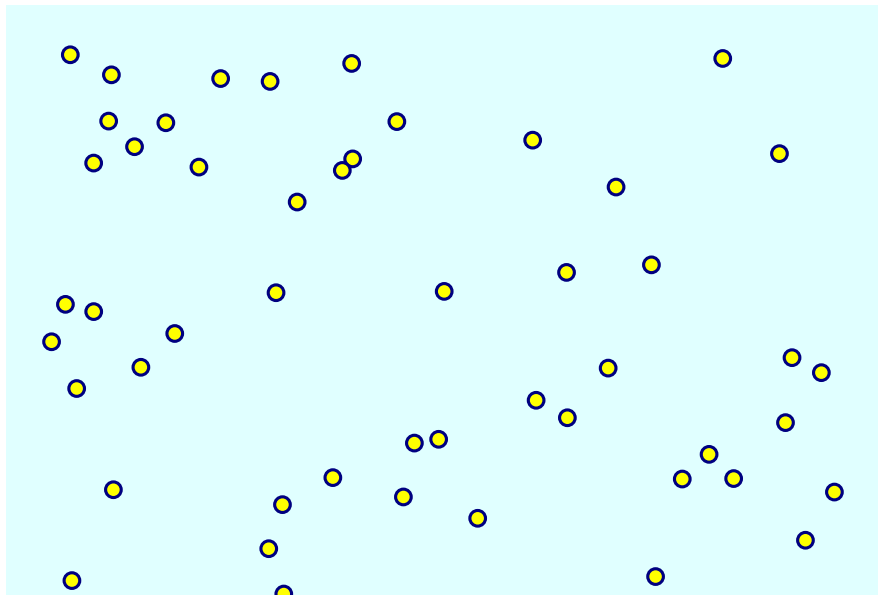
- 1 For any indices $1 \leq i_1 < i_2 < \dots < i_k \leq n$, and observe that $\Pr[\mathcal{E}_{i_k} \mid \mathcal{E}_{i_1} \cap \dots \cap \mathcal{E}_{i_{k-1}}] = \Pr[\mathcal{E}_{i_k}]$,
- 2 ..because \mathcal{E}_{i_1} determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_{i_k}$.
- 3 By induction: $\Pr[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1} \mid \mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] \Pr[\mathcal{E}_{i_2} \cap \dots \cap \mathcal{E}_{i_k}] = \Pr[\mathcal{E}_{i_1}] \Pr[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \dots \cap \mathcal{E}_{i_k}] = \prod_{j=1}^k \Pr[\mathcal{E}_{i_j}] = \prod_{j=1}^k \frac{1}{i_j}$.
- 4 \implies variables X_1, \dots, X_n are independent.
- 5 Result readily follows from Chernoff's inequality. ■

13.3: Closest pair in linear time

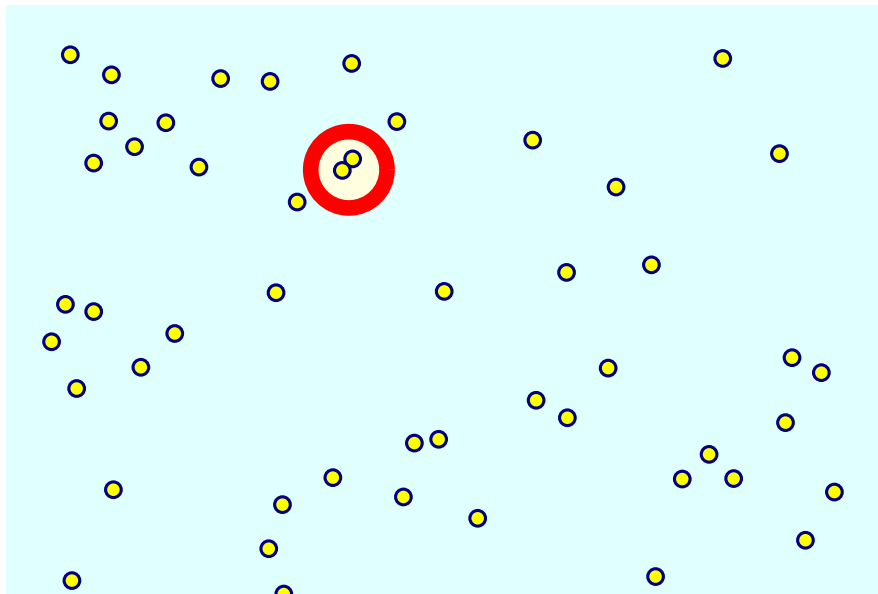
Finding the closest pair of points



Finding the closest pair of points



Finding the closest pair of points

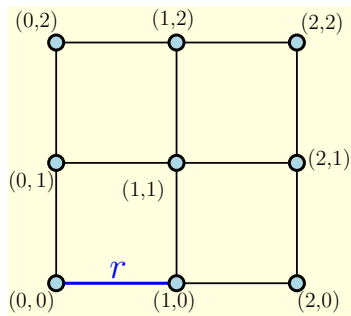


Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.
- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.



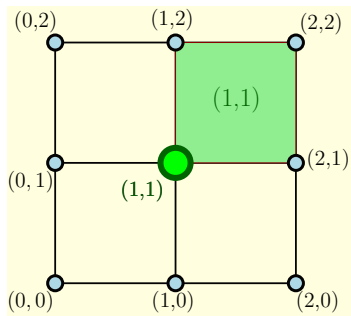
Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.

- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.

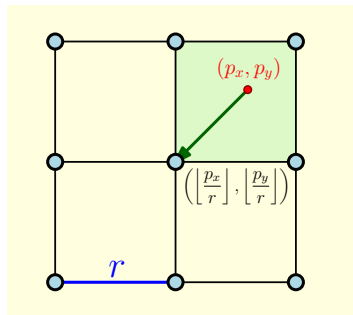


Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.
- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.

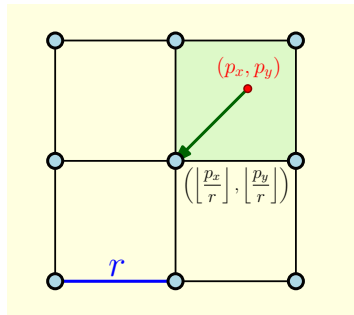


Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.
- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.

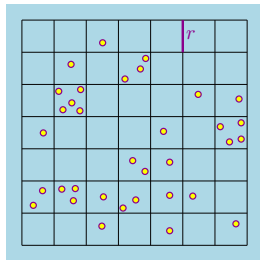
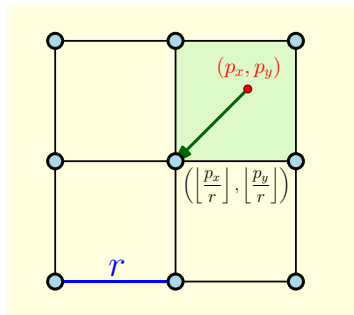


Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.
- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.

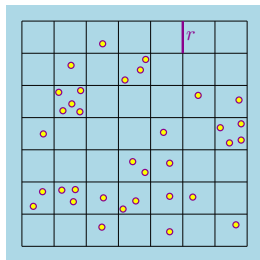
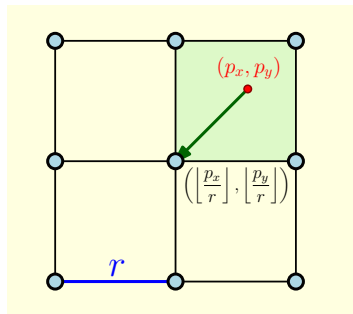


Grids...

- 1 r : Side length of grid cell.
- 2 Grid cell IDed by pair of integers.
- 3 Constant time to determine a point p 's grid cell id:

$$id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)$$

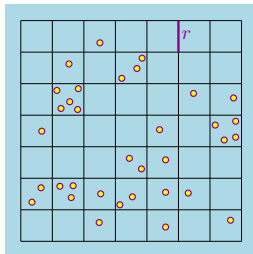
- 4 Limited use of the floor function (but no word packing tricks).
- 5 Use hashing on (grid) points.
- 6 Store points in grid...
...in linear time.



Storing point set in grid/hash-table...

Hashing:

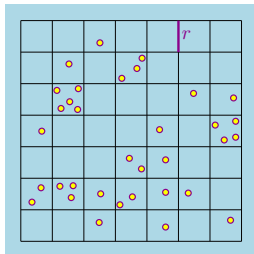
- 1 Non-empty grid cells
- 2 For non-empty grid cell:
List of points in it.
- 3 For a grid cell:
Its neighboring cells.



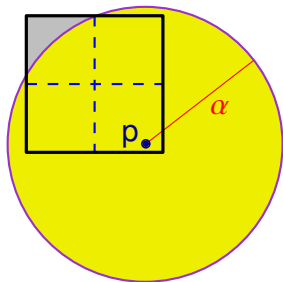
Storing point set in grid/hash-table...

Hashing:

- 1 Non-empty grid cells
- 2 For non-empty grid cell:
List of points in it.
- 3 For a grid cell:
Its neighboring cells.



Closet pair in a square



Lemma

Let \mathbf{P} be a set of points contained inside a square \square , such that the sidelength of \square is $\alpha = \mathcal{CP}(\mathbf{P})$. Then $|\mathbf{P}| \leq 4$.

Proof.

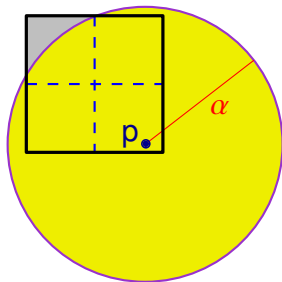
Partition \square into four equal squares $\square_1, \dots, \square_4$.

Each square diameter $\sqrt{2}\alpha/2 < \alpha$.

... contain at most one point of \mathbf{P} ; that is, the disk of radius α centered at a point $\mathbf{p} \in \mathbf{P}$ completely covers the subsquare containing it; see the figure on the right.

\mathbf{P} can have four points if it is the four corners of \square . □

Closet pair in a square



Lemma

Let \mathbf{P} be a set of points contained inside a square \square , such that the sidelength of \square is $\alpha = \mathcal{CP}(\mathbf{P})$. Then $|\mathbf{P}| \leq 4$.

Proof.

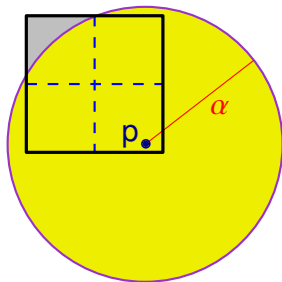
Partition \square into four equal squares $\square_1, \dots, \square_4$.

Each square diameter $\sqrt{2}\alpha/2 < \alpha$.

... contain at most one point of \mathbf{P} ; that is, the disk of radius α centered at a point $\mathbf{p} \in \mathbf{P}$ completely covers the subsquare containing it; see the figure on the right.

\mathbf{P} can have four points if it is the four corners of \square . □

Closet pair in a square



Lemma

Let \mathbf{P} be a set of points contained inside a square \square , such that the sidelength of \square is $\alpha = \mathcal{CP}(\mathbf{P})$. Then $|\mathbf{P}| \leq 4$.

Proof.

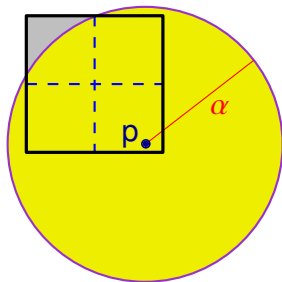
Partition \square into four equal squares $\square_1, \dots, \square_4$.

Each square diameter $\sqrt{2}\alpha/2 < \alpha$.

... contain at most one point of \mathbf{P} ; that is, the disk of radius α centered at a point $\mathbf{p} \in \mathbf{P}$ completely covers the subsquare containing it; see the figure on the right.

\mathbf{P} can have four points if it is the four corners of \square . □

Closest pair in a square



Lemma

Let \mathbf{P} be a set of points contained inside a square \square , such that the sidelength of \square is $\alpha = \mathcal{CP}(\mathbf{P})$. Then $|\mathbf{P}| \leq 4$.

Proof.

Partition \square into four equal squares $\square_1, \dots, \square_4$.

Each square diameter $\sqrt{2}\alpha/2 < \alpha$.

... contain at most one point of \mathbf{P} ; that is, the disk of radius α centered at a point $\mathbf{p} \in \mathbf{P}$ completely covers the subsquare containing it; see the figure on the right.

\mathbf{P} can have four points if it is the four corners of \square . □

Verify closet pair

Lemma

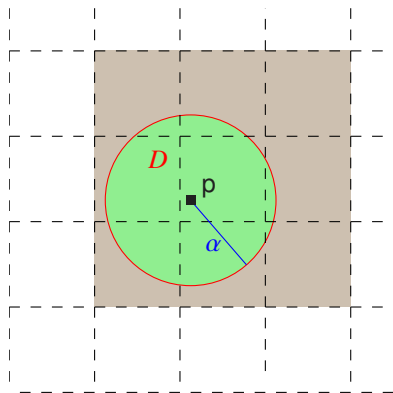
P: set of n points in the plane. α : distance. Verify in linear time whether $\mathcal{CP}(\mathbf{P}) < \alpha$, $\mathcal{CP}(\mathbf{P}) = \alpha$, or $\mathcal{CP}(\mathbf{P}) > \alpha$.

proof

Indeed, store the points of **P** in the grid \mathbf{G}_α . For every non-empty grid cell, we maintain a linked list of the points inside it. Thus, adding a new point **p** takes constant time. Specifically, compute $\text{id}(\mathbf{p})$, check if $\text{id}(\mathbf{p})$ already appears in the hash table, if not, create a new linked list for the cell with this ID number, and store **p** in it. If a linked list already exists for $\text{id}(\mathbf{p})$, just add **p** to it. This takes $O(n)$ time overall.

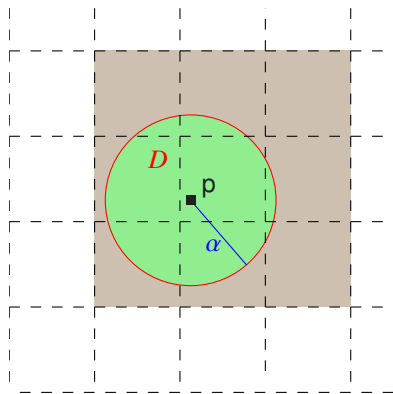
Now, if any grid cell in $\mathbf{G}_\alpha(\mathbf{P})$ contains more than, say, 4 points of **P**, then it must be that the $\mathcal{CP}(\mathbf{P}) < \alpha$, by previous lemma.

Proof continued



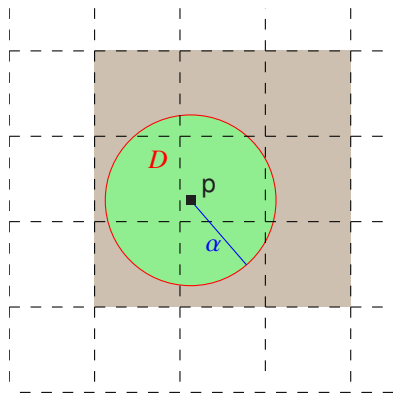
- 1 When insert a point \mathbf{p} : fetch all the points of \mathbf{P} in cluster of \mathbf{P}
- 2 Takes constant time.
- 3 If there is a point closer to \mathbf{p} than α that was already inserted, then it must be stored in one of these 9 cells.
- 4 Now, each one of those cells must contain at most 4 points of \mathbf{P} by prev lemma.
- 5 Otherwise, already stopped since $\mathcal{CP}(\cdot) < \alpha$.

Proof continued



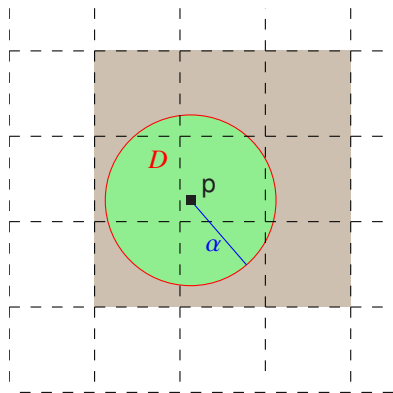
- 1 When insert a point \mathbf{p} : fetch all the points of \mathbf{P} in cluster of \mathbf{P}
- 2 Takes constant time.
- 3 If there is a point closer to \mathbf{p} than α that was already inserted, then it must be stored in one of these 9 cells.
- 4 Now, each one of those cells must contain at most 4 points of \mathbf{P} by prev lemma.
- 5 Otherwise, already stopped since $\mathcal{CP}(\cdot) < \alpha$.

Proof continued



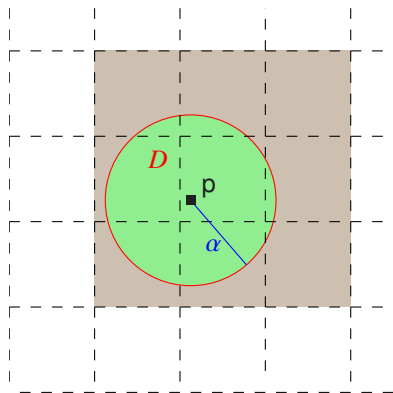
- 1 When insert a point \mathbf{p} : fetch all the points of \mathbf{P} in cluster of \mathbf{P}
- 2 Takes constant time.
- 3 If there is a point closer to \mathbf{p} than α that was already inserted, then it must be stored in one of these 9 cells.
- 4 Now, each one of those cells must contain at most 4 points of \mathbf{P} by prev lemma.
- 5 Otherwise, already stopped since $\mathcal{CP}(\cdot) < \alpha$.

Proof continued



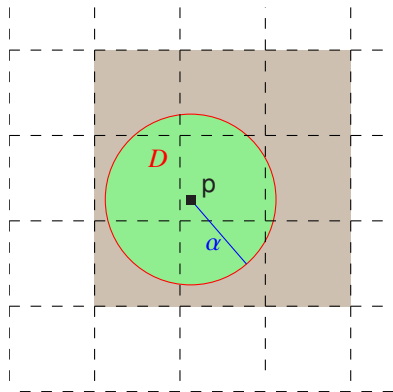
- 1 When insert a point \mathbf{p} : fetch all the points of \mathbf{P} in cluster of \mathbf{P}
- 2 Takes constant time.
- 3 If there is a point closer to \mathbf{p} than α that was already inserted, then it must be stored in one of these $\mathbf{9}$ cells.
- 4 Now, each one of those cells must contain at most $\mathbf{4}$ points of \mathbf{P} by prev lemma.
- 5 Otherwise, already stopped since $\mathcal{CP}(\cdot) < \alpha$.

Proof continued



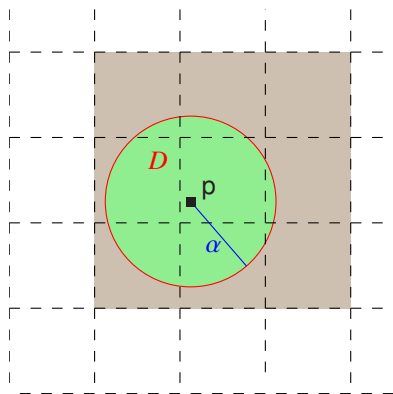
- 1 When insert a point \mathbf{p} : fetch all the points of \mathbf{P} in cluster of \mathbf{P}
- 2 Takes constant time.
- 3 If there is a point closer to \mathbf{p} than α that was already inserted, then it must be stored in one of these 9 cells.
- 4 Now, each one of those cells must contain at most 4 points of \mathbf{P} by prev lemma.
- 5 Otherwise, already stopped since $\mathcal{CP}(\cdot) < \alpha$.

Proof continued



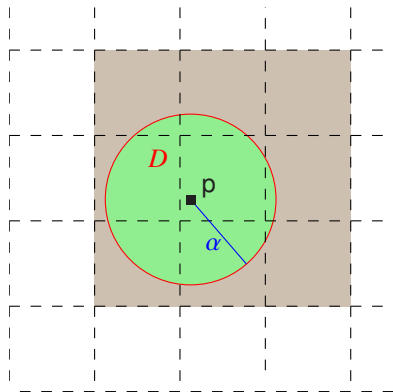
- 1 S set of all points in cluster.
- 2 $|S| \leq 9 \cdot 4 = O(1)$.
- 3 Compute closest point to \mathbf{p} in S . $O(1)$ time.
- 4 If $d(\mathbf{p}, S) < \alpha$, we stop; otherwise, continue to next point.
- 5 Correctness: ' $\mathcal{CP}(\mathbf{P}) < \alpha$ ' returned only if such pair found.

Proof continued



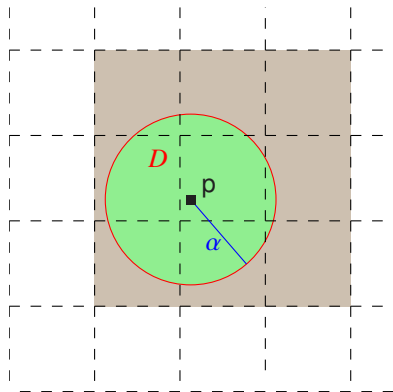
- 1 S set of all points in cluster.
- 2 $|S| \leq 9 \cdot 4 = O(1)$.
- 3 Compute closest point to \mathbf{p} in S . $O(1)$ time.
- 4 If $d(\mathbf{p}, S) < \alpha$, we stop; otherwise, continue to next point.
- 5 Correctness: ' $\mathcal{CP}(\mathbf{P}) < \alpha$ ' returned only if such pair found.

Proof continued



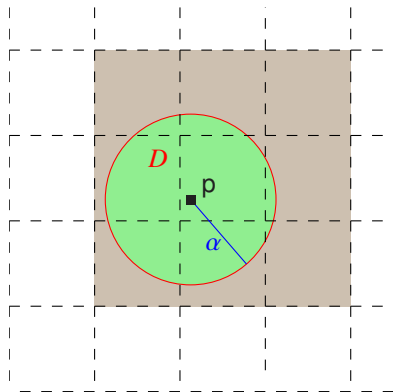
- 1 S set of all points in cluster.
- 2 $|S| \leq 9 \cdot 4 = O(1)$.
- 3 Compute closest point to \mathbf{p} in S . $O(1)$ time.
- 4 If $d(\mathbf{p}, S) < \alpha$, we stop; otherwise, continue to next point.
- 5 Correctness: ' $\mathcal{CP}(\mathbf{P}) < \alpha$ ' returned only if such pair found.

Proof continued



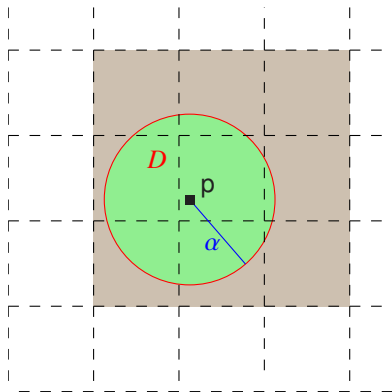
- 1 S set of all points in cluster.
- 2 $|S| \leq 9 \cdot 4 = O(1)$.
- 3 Compute closest point to \mathbf{p} in S . $O(1)$ time.
- 4 If $d(\mathbf{p}, S) < \alpha$, we stop; otherwise, continue to next point.
- 5 Correctness: ' $\mathcal{CP}(\mathbf{P}) < \alpha$ ' returned only if such pair found.

Proof continued



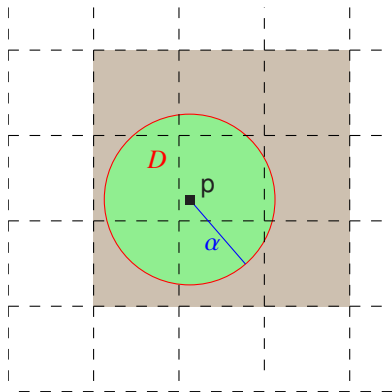
- 1 S set of all points in cluster.
- 2 $|S| \leq 9 \cdot 4 = O(1)$.
- 3 Compute closest point to \mathbf{p} in S . $O(1)$ time.
- 4 If $d(\mathbf{p}, S) < \alpha$, we stop; otherwise, continue to next point.
- 5 Correctness: ' $\mathcal{CP}(\mathbf{P}) < \alpha$ ' returned only if such pair found.

Proof continued



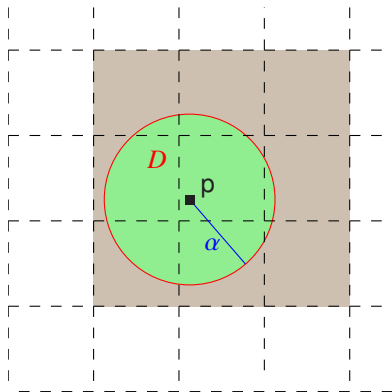
- 1 Assume \mathbf{p} and \mathbf{q} : realizing closest pair.
- 2 $\|\mathbf{p} - \mathbf{q}\| = \mathcal{CP}(\mathbf{P}) < \alpha$.
- 3 When later point (say \mathbf{p}) inserted, the set \mathcal{S} would contain \mathbf{q} .
- 4 algorithm would stop and return ' $\mathcal{CP}(\mathbf{P}) < \alpha$ '.
- 5 ■

Proof continued



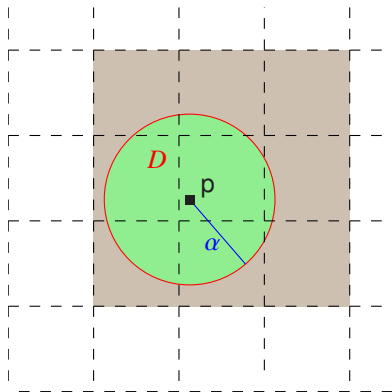
- 1 Assume \mathbf{p} and \mathbf{q} : realizing closest pair.
- 2 $\|\mathbf{p} - \mathbf{q}\| = \mathcal{CP}(\mathbf{P}) < \alpha$.
- 3 When later point (say \mathbf{p}) inserted, the set \mathcal{S} would contain \mathbf{q} .
- 4 algorithm would stop and return ' $\mathcal{CP}(\mathbf{P}) < \alpha$ '.
- 5 ■

Proof continued



- 1 Assume \mathbf{p} and \mathbf{q} : realizing closest pair.
- 2 $\|\mathbf{p} - \mathbf{q}\| = \mathcal{CP}(\mathbf{P}) < \alpha$.
- 3 When later point (say \mathbf{p}) inserted, the set \mathcal{S} would contain \mathbf{q} .
- 4 algorithm would stop and return ' $\mathcal{CP}(\mathbf{P}) < \alpha$ '.
- 5 ■

Proof continued

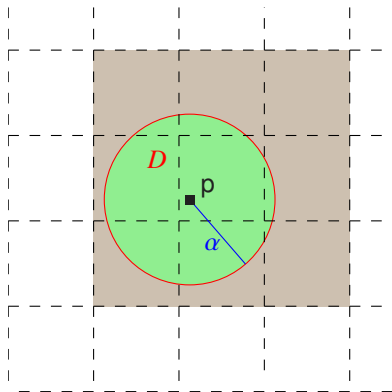


- 1 Assume \mathbf{p} and \mathbf{q} : realizing closest pair.
- 2 $\|\mathbf{p} - \mathbf{q}\| = \mathcal{CP}(\mathbf{P}) < \alpha$.
- 3 When later point (say \mathbf{p}) inserted, the set \mathcal{S} would contain \mathbf{q} .
- 4 algorithm would stop and return ' $\mathcal{CP}(\mathbf{P}) < \alpha$ '.

5



Proof continued



- 1 Assume \mathbf{p} and \mathbf{q} : realizing closest pair.
- 2 $\|\mathbf{p} - \mathbf{q}\| = \mathcal{CP}(\mathbf{P}) < \alpha$.
- 3 When later point (say \mathbf{p}) inserted, the set \mathcal{S} would contain \mathbf{q} .
- 4 algorithm would stop and return ' $\mathcal{CP}(\mathbf{P}) < \alpha$ '.
- 5 ■

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closet pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closet pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

New algorithm

- 1 Pick a random permutation of the points of \mathbf{P} .
- 2 $\langle \mathbf{p}_1, \dots, \mathbf{p}_n \rangle$ be this permutation.
- 3 $\alpha_2 = \|\mathbf{p}_1 - \mathbf{p}_2\|$.
- 4 Insert points into the closet-pair distance verifying data-structure.
- 5 α_i : the closest pair distance in the set $\mathbf{P}_i = \{\mathbf{p}_1, \dots, \mathbf{p}_i\}$, for $i = 2, \dots, n$.
- 6 i th iteration:
 - 1 if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
 - 2 If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i .
 - 3 rebuild the grid, and reinsert the i points of \mathbf{P}_i from scratch into the grid \mathbf{G}_{α_i} . Takes $O(i)$ time.
- 7 Returns the number α_n and points realizing it.

Weak analysis...

Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix P_i .
- 5 $q \in P_i$ is critical if $\mathcal{CP}(P_i \setminus \{q\}) > \mathcal{CP}(P_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.

Weak analysis...

Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix P_i .
- 5 $q \in P_i$ is critical if $\mathcal{CP}(P_i \setminus \{q\}) > \mathcal{CP}(P_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.

Weak analysis...

Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix P_i .
- 5 $q \in P_i$ is critical if $\mathcal{CP}(P_i \setminus \{q\}) > \mathcal{CP}(P_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.

Weak analysis...

Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix \mathbf{P}_i .
- 5 $q \in \mathbf{P}_i$ is critical if $\mathcal{CP}(\mathbf{P}_i \setminus \{q\}) > \mathcal{CP}(\mathbf{P}_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.

Weak analysis...

Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.

proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix \mathbf{P}_i .
- 5 $\mathbf{q} \in \mathbf{P}_i$ is **critical** if $\mathcal{CP}(\mathbf{P}_i \setminus \{\mathbf{q}\}) > \mathcal{CP}(\mathbf{P}_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.

Weak analysis...


Lemma

Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \dots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.


proof

- 1 $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
- 2 $\mathbf{E}[X_i] = \Pr[X_i = 1]$ and $t = \sum_{i=3}^n X_i$.
- 3 $\Pr[X_i = 1] = \Pr[\alpha_i < \alpha_{i-1}]$.
- 4 Backward analysis. Fix \mathbf{P}_i .
- 5 $\mathbf{q} \in \mathbf{P}_i$ is **critical** if $\mathcal{CP}(\mathbf{P}_i \setminus \{\mathbf{q}\}) > \mathcal{CP}(\mathbf{P}_i)$.
- 6 No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\Pr[X_i = 1] = 0$.


Proof continued...

- 1 If one critical point, then $\Pr[X_i = 1] = 1/i$.
- 2 Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- 3 $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- 4 $\Pr[X_i = 1] = 2/i$.
- 5 Cannot be more than two critical points.
- 6 Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.
- 7 


Proof continued...

- 1 If one critical point, then $\Pr[X_i = 1] = 1/i$.
- 2 Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- 3 $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- 4 $\Pr[X_i = 1] = 2/i$.
- 5 Cannot be more than two critical points.
- 6 Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.
- 7 

Proof continued...

- 1 If one critical point, then $\Pr[X_i = 1] = 1/i$.
- 2 Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- 3 $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- 4 $\Pr[X_i = 1] = 2/i$.
- 5 Cannot be more than two critical points.
- 6 Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.
- 7 

Proof continued...

- 1 If one critical point, then $\Pr[X_i = 1] = 1/i$.
- 2 Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- 3 $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- 4 $\Pr[X_i = 1] = 2/i$.
- 5 Cannot be more than two critical points.
- 6 Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.
- 7 

Proof continued...

- 1 If one critical point, then $\Pr[X_i = 1] = 1/i$.
- 2 Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- 3 $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- 4 $\Pr[X_i = 1] = 2/i$.
- 5 Cannot be more than two critical points.
- 6 Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.

7



Proof continued...

- ① If one critical point, then $\Pr[X_i = 1] = 1/i$.
- ② Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- ③ $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- ④ $\Pr[X_i = 1] = 2/i$.
- ⑤ Cannot be more than two critical points.
- ⑥ Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.

7



Proof continued...

- ① If one critical point, then $\Pr[X_i = 1] = 1/i$.
- ② Assume two critical points and let \mathbf{p}, \mathbf{q} be this unique pair of points of \mathbf{P}_i realizing $\mathcal{CP}(\mathbf{P}_i)$.
- ③ $\alpha_i < \alpha_{i-1} \iff \mathbf{p}$ or \mathbf{q} is \mathbf{p}_i .
- ④ $\Pr[X_i = 1] = 2/i$.
- ⑤ Cannot be more than two critical points.
- ⑥ Linearity of expectations: $\mathbf{E}[t] = \mathbf{E}[\sum_{i=3}^n X_i] = \sum_{i=3}^n \mathbf{E}[X_i] \leq \sum_{i=3}^n 2/i = O(\log n)$.
- ⑦ ■

Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- ① $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- ② Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- ③ $E[R] = E[1 + \sum_{i=3}^n (1 + X_i \cdot i)] \leq n + \sum_{i=3}^n E[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $E[X_i] = \Pr[X_i = 1] \leq 2/i$.
- ④ Expected running time of the algorithm is $O(E[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- 1 $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- 2 Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- 3 $E[R] = E[1 + \sum_{i=3}^n (1 + X_i \cdot i)] \leq n + \sum_{i=3}^n E[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $E[X_i] = \Pr[X_i = 1] \leq 2/i$.
- 4 Expected running time of the algorithm is $O(E[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- 1 $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- 2 Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- 3 $\mathbf{E}[R] = \mathbf{E}\left[1 + \sum_{i=3}^n (1 + X_i \cdot i)\right] \leq n + \sum_{i=3}^n \mathbf{E}[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $\mathbf{E}[X_i] = \Pr[X_i = 1] \leq 2/i$.
- 4 Expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- ① $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- ② Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- ③ $\mathbf{E}[R] = \mathbf{E}\left[1 + \sum_{i=3}^n (1 + X_i \cdot i)\right] \leq n + \sum_{i=3}^n \mathbf{E}[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $\mathbf{E}[X_i] = \Pr[X_i = 1] \leq 2/i$.
- ④ Expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- 1 $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- 2 Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- 3 $\mathbf{E}[R] = \mathbf{E}\left[1 + \sum_{i=3}^n (1 + X_i \cdot i)\right] \leq n + \sum_{i=3}^n \mathbf{E}[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $\mathbf{E}[X_i] = \Pr[X_i = 1] \leq 2/i$.
- 4 Expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- ① $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- ② Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- ③ $\mathbf{E}[R] = \mathbf{E}\left[1 + \sum_{i=3}^n (1 + X_i \cdot i)\right] \leq n + \sum_{i=3}^n \mathbf{E}[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $\mathbf{E}[X_i] = \Pr[X_i = 1] \leq 2/i$.
- ④ Expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■



Expected linear time analysis...

Theorem

P: set of n points in the plane. Compute the closest pair of **P** in expected linear time.

Proof.

- ① $X_i = 1 \iff \alpha_i \neq \alpha_{i-1}$.
- ② Running time is proportional to $R = 1 + \sum_{i=3}^n (1 + X_i \cdot i)$.
- ③ $\mathbf{E}[R] = \mathbf{E}\left[1 + \sum_{i=3}^n (1 + X_i \cdot i)\right] \leq n + \sum_{i=3}^n \mathbf{E}[X_i] \cdot i \leq n + \sum_{i=3}^n i \cdot \Pr[X_i = 1] \leq n + \sum_{i=3}^n i \cdot \frac{2}{i} \leq 3n$, by linearity of expectation and since $\mathbf{E}[X_i] = \Pr[X_i = 1] \leq 2/i$.
- ④ Expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■



13.4: Computing nets

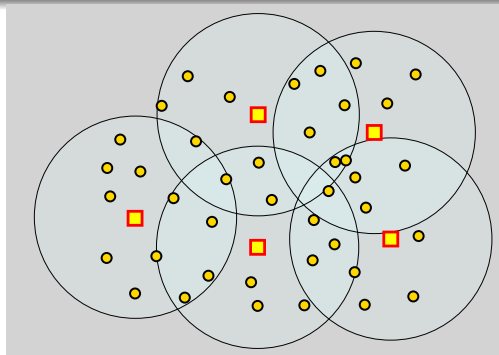
Nets

The Main Tool

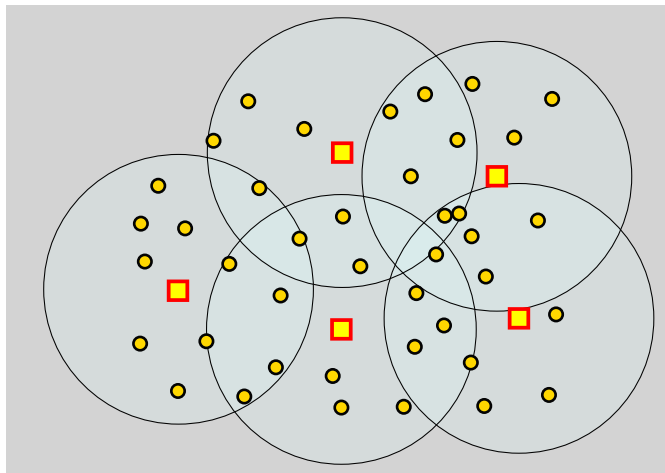
r -net

$N \subseteq P$ is an r -net if

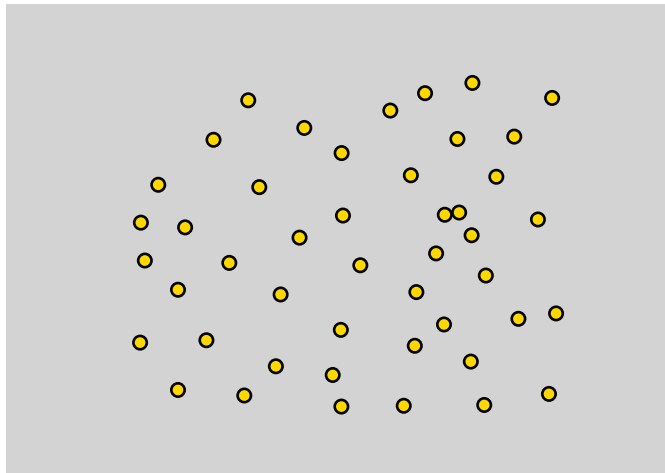
- Every point in P has distance $< r$ to a point in N
- For any two $p, q \in N$, we have $d(p, q) \geq r$.



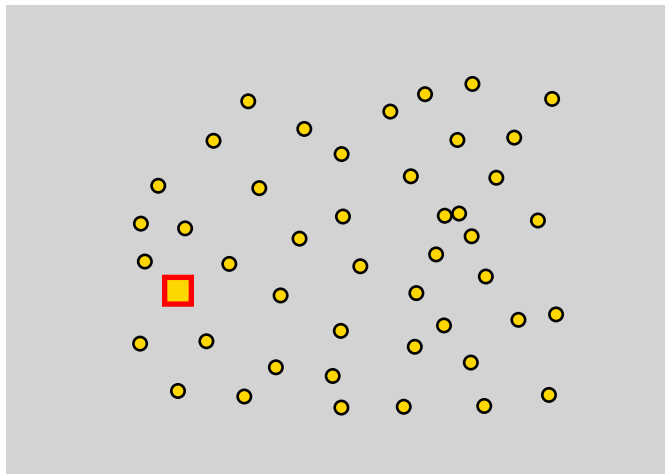
Computing an r-net



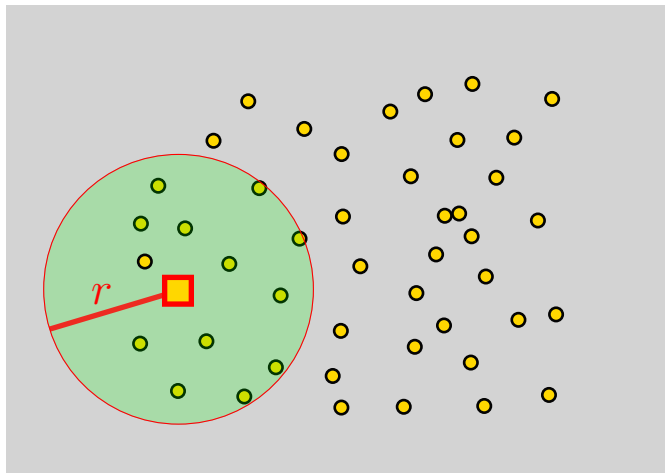
Computing an r-net



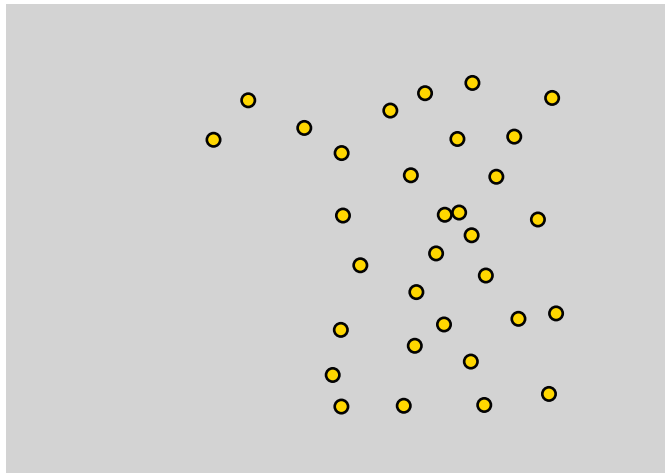
Computing an r-net



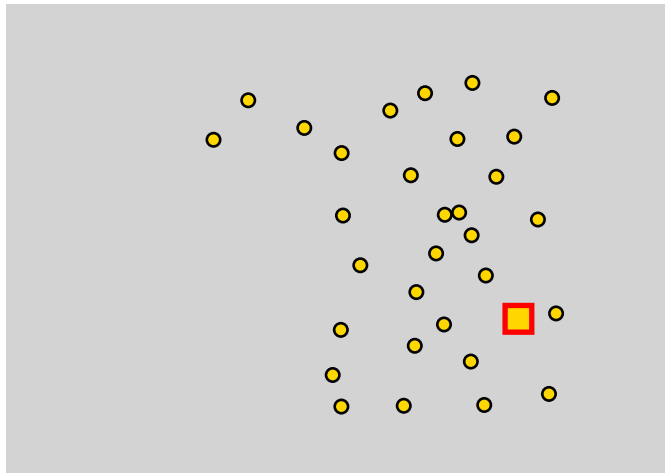
Computing an r-net



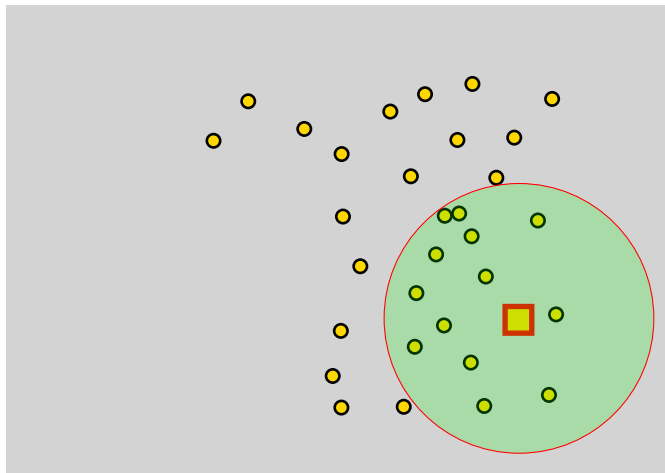
Computing an r-net



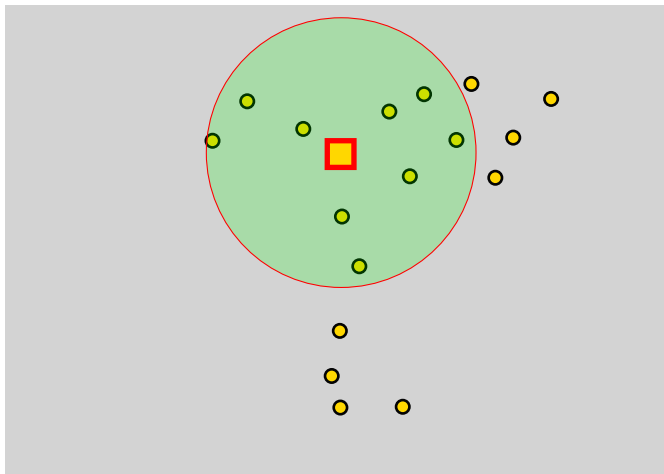
Computing an r-net



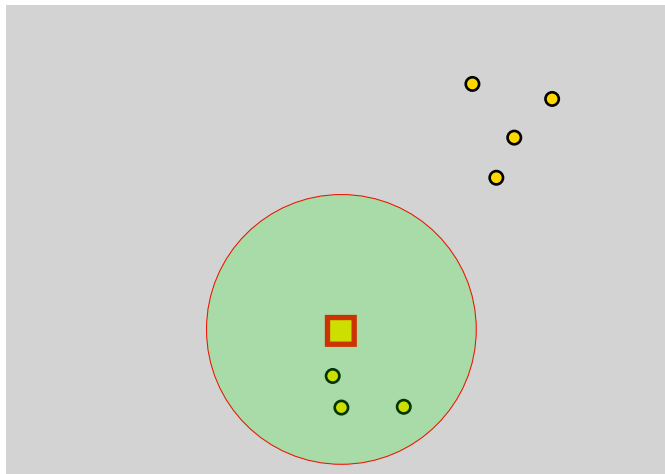
Computing an r-net



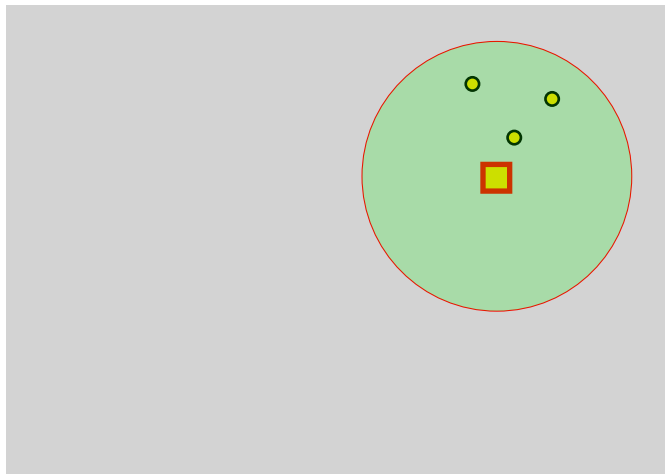
Computing an r-net



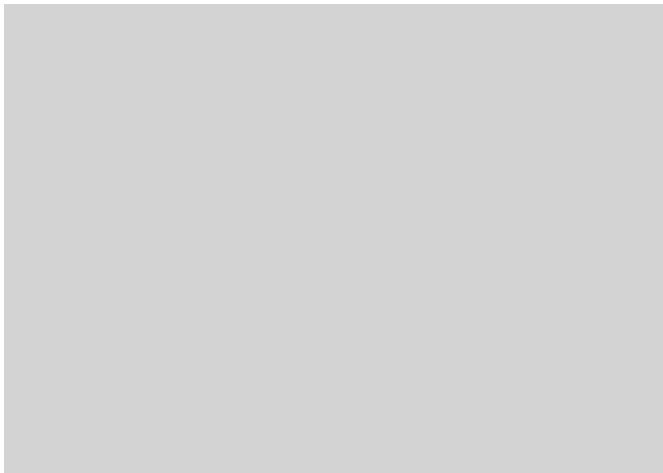
Computing an r-net



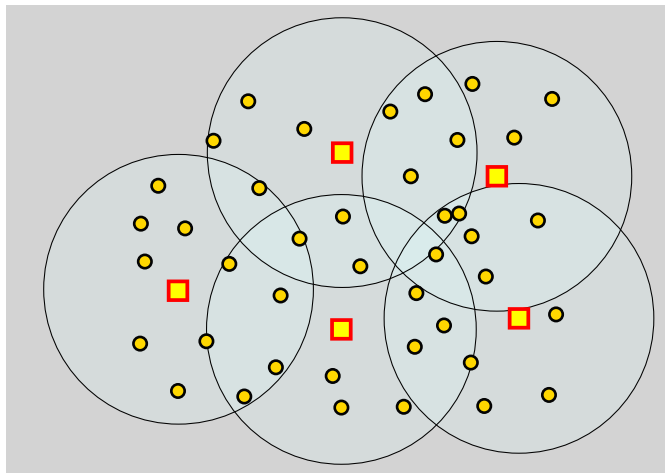
Computing an r-net



Computing an r-net

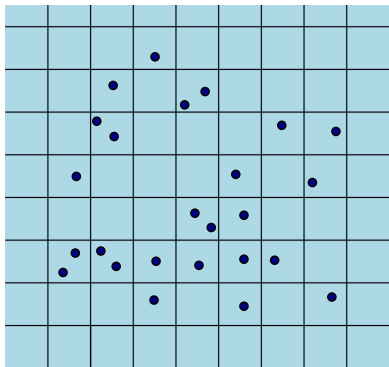


Computing an r-net



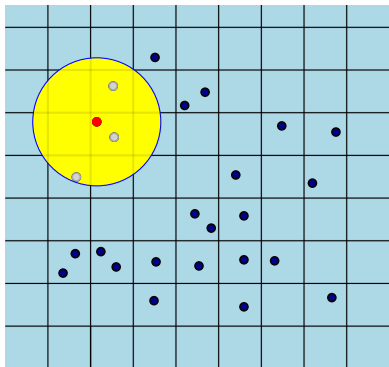
Application of Grids: Computing nets

...in linear time



Application of Grids: Computing nets

...in linear time



Repeatedly:

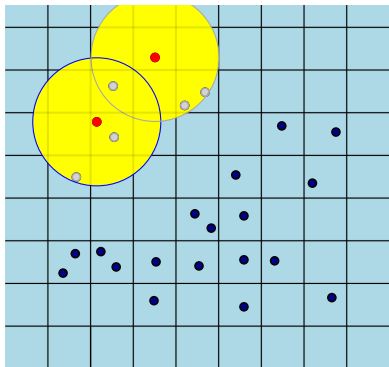
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

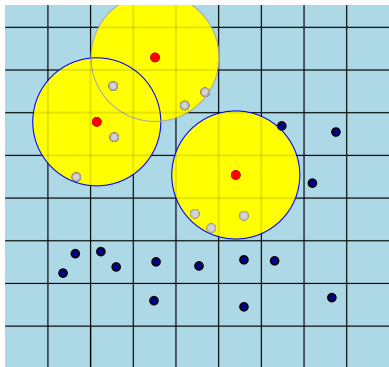
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

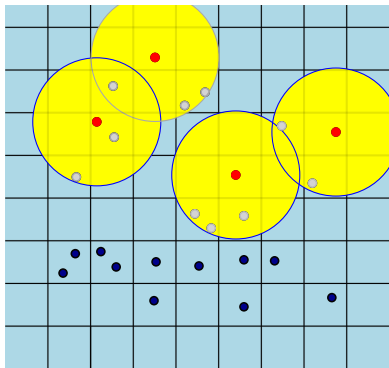
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

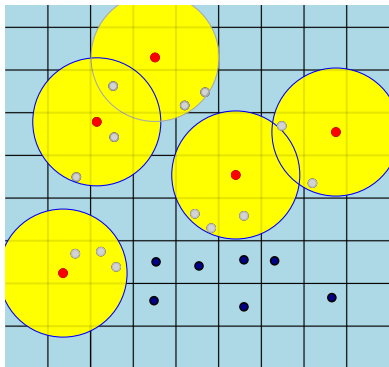
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

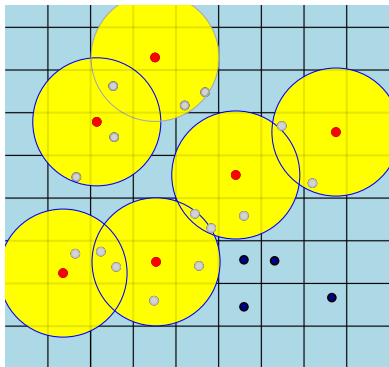
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

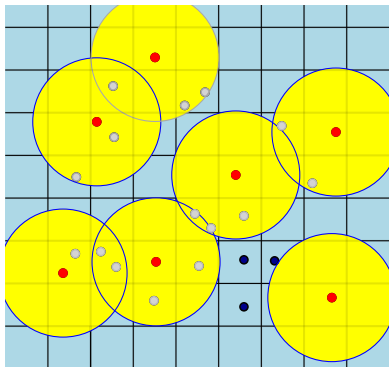
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

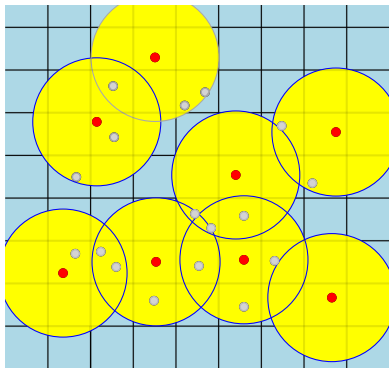
- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

Application of Grids: Computing nets

...in linear time



Repeatedly:

- (1) Pick any unmarked point.
- (2) Mark all neighbors in distance $< r$.

In an r -grid

- (A) Neighbors in distance $< r$, are in neighboring cells.
- (B) Neighboring Cells found in $O(1)$ time.
- (C) Cells contain lists of points.

13.5: Computing a good ordering of the vertices of a graph

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Input

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$: edge-weighted.
- 2 n vertices and m edges.
- 3 Task: compute an ordering $\pi = \langle \pi_1, \dots, \pi_n \rangle$ of vertices.
- 4 $\forall v \in \mathbf{V} \quad L_v :$
 $\pi_i \in L_v, \iff \pi_i$ closet vertex to v in prefix $\langle \pi_1, \dots, \pi_i \rangle$.
- 5 Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
- 6 ... client needs to maintain its current closest server.
- 7 How min total size of lists? $\mathcal{L} = \sum_{v \in \mathbf{V}} |L_v|$.

Algorithm

- ① π_1, \dots, π_n : random permutation of \mathbf{V} of \mathbf{G} .
- ② $\forall v \in \mathbf{V} \quad \delta(v) = +\infty$.
- ③ For $i = 1$ to n do:
 - ① $\delta(\pi_i)$ to 0 ,
 - ② start Dijkstra from the i th vertex π_i .
 - ③ Dijkstra propagates to u only if improves current distance.
 - ④ Update $\delta(u)$ to $\mathbf{d}_{\mathbf{G}}(\pi_i, u) \iff \mathbf{d}_{\mathbf{G}}(\pi_i, u) < \delta'(u)$
 $\delta'(u)$: value before this iteration started.
 - ⑤ If $\delta(u)$ updated: add π_i to L_u .

13.5.1: Analysis

Lemma

Algorithm computes a permutation π , such that:

- 1 $\mathbf{E}[|\mathcal{L}|] = O(n \log n)$.
- 2 Expected running time $O((n \log n + m) \log n)$.
- 3 $n = |\mathbf{V}(\mathbf{G})|$ and $m = |\mathbf{E}(\mathbf{G})|$.
- 4 Both bounds also hold with high probability.

Proof

- 1 Fix a vertex $v \in V = \{v_1, \dots, v_n\}$.
- 2 $U = \{d_G(v, v_1), \dots, d_G(v, v_n)\}$.
- 3 $d_G(v, \pi_1), \dots, d_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in E(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 time(**decrease-key**)= $O(1)$ time(**delete-min**)= $O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in \mathbf{V} = \{v_1, \dots, v_n\}$.
- 2 $U = \{\mathbf{d}_G(v, v_1), \dots, \mathbf{d}_G(v, v_n)\}$.
- 3 $\mathbf{d}_G(v, \pi_1), \dots, \mathbf{d}_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in \mathbf{E}(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 time(**decrease-key**)= $O(1)$ time(**delete-min**)= $O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in \mathbf{V} = \{v_1, \dots, v_n\}$.
- 2 $U = \{\mathbf{d}_G(v, v_1), \dots, \mathbf{d}_G(v, v_n)\}$.
- 3 $\mathbf{d}_G(v, \pi_1), \dots, \mathbf{d}_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in \mathbf{E}(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 time(**decrease-key**)= $O(1)$ time(**delete-min**)= $O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in \mathbf{V} = \{v_1, \dots, v_n\}$.
- 2 $U = \{\mathbf{d}_G(v, v_1), \dots, \mathbf{d}_G(v, v_n)\}$.
- 3 $\mathbf{d}_G(v, \pi_1), \dots, \mathbf{d}_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in \mathbf{E}(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 time(**decrease-key**)= $O(1)$ time(**delete-min**)= $O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in V = \{v_1, \dots, v_n\}$.
- 2 $U = \{d_G(v, v_1), \dots, d_G(v, v_n)\}$.
- 3 $d_G(v, \pi_1), \dots, d_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in E(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 time(**decrease-key**)= $O(1)$ time(**delete-min**)= $O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in \mathbf{V} = \{v_1, \dots, v_n\}$.
- 2 $U = \{\mathbf{d}_G(v, v_1), \dots, \mathbf{d}_G(v, v_n)\}$.
- 3 $\mathbf{d}_G(v, \pi_1), \dots, \mathbf{d}_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in \mathbf{E}(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 $\text{time}(\mathbf{decrease-key}) = O(1)$ $\text{time}(\mathbf{delete-min}) = O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in V = \{v_1, \dots, v_n\}$.
- 2 $U = \{d_G(v, v_1), \dots, d_G(v, v_n)\}$.
- 3 $d_G(v, \pi_1), \dots, d_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in E(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 $\text{time}(\text{decrease-key}) = O(1)$ $\text{time}(\text{delete-min}) = O(\log n)$.
(Fibonacci heaps).

Proof

- 1 Fix a vertex $v \in V = \{v_1, \dots, v_n\}$.
- 2 $U = \{d_G(v, v_1), \dots, d_G(v, v_n)\}$.
- 3 $d_G(v, \pi_1), \dots, d_G(v, \pi_n)$: random permutation of U .
- 4 By lemma seen π min changes $O(\log n)$ times in expectations + high prob.
- 5 $|L_v| = O(\log n)$ in expectation + high probability.
- 6 Running time:
 - 1 For $uv \in E(G)$: $\delta(u)$ or $\delta(v)$ changes $O(\log n)$ times.
 - 2 uv gets visited $O(\log n)$ times by all “Disjkstras”,
 - 3 Overall running time $O(n \log^2 n + m \log n)$:
 - 1 $O(n \log n)$ changes in $\delta(\cdot)$.
 - 2 n : **delete-min** operations
 - 3 Edge triggers $O(\log n)$ **decrease-key** operations.
 - 4 $\text{time}(\text{decrease-key}) = O(1)$ $\text{time}(\text{delete-min}) = O(\log n)$.
(Fibonacci heaps).

13.6: Computing an r -net in a sparse graph

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Computing r -net in sparse graphs.

- 1 $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a weighted graph with n vertices and m edges, let $r > 0$.
- 2 π_i : i th vertex in a random permutation of \mathbf{V} .
- 3 $\forall v \in \mathbf{V} : \delta(v) := +\infty$.
- 4 Test whether $\delta(\pi_i) \geq r$, if so:
 - 1 Add π_i to the resulting net \mathcal{N} .
 - 2 Set $\delta(\pi_i)$ to zero.
 - 3 Perform Dijkstra's algorithm starting from π_i ,
Occupy a vertex u only if improve distance: $\delta(u)$.
 - 4 If a vertex u is expanded: $\delta(u)$: computed distance from π_i ,
and relax the edges adjacent to u in the graph.

Correctness

Lemma

The set \mathcal{N} is an r -net in \mathbf{G} .

Proof.

- 1 End: $\forall v \in \mathbf{V}: \delta(v) < r$.
- 2 By induction: if $\ell = \delta(v)$, for some vertex v , then the distance of v to the set \mathcal{N} is at most ℓ .
- 3 Every two points in \mathcal{N} have distance $\geq r$. Indeed, when the algorithm handles vertex $v \in \mathcal{N}$, its distance from all the vertices currently in \mathcal{N} is $\geq r$.



Lemma

The set \mathcal{N} is an r -net in \mathbf{G} .

Proof.

- 1 End: $\forall v \in \mathbf{V}: \delta(v) < r$.
- 2 By induction: if $\ell = \delta(v)$, for some vertex v , then the distance of v to the set \mathcal{N} is at most ℓ .
- 3 Every two points in \mathcal{N} have distance $\geq r$. Indeed, when the algorithm handles vertex $v \in \mathcal{N}$, its distance from all the vertices currently in \mathcal{N} is $\geq r$.



Lemma

Consider an execution of the algorithm, and any vertex $v \in \mathbf{V}$. The expected number of times the algorithm updates the value of $\delta(v)$ during its execution is $O(\log n)$, and more strongly the number of updates is $O(\log n)$ with high probability.

Proof...

- 1 Assume all distances in \mathbf{G} are distinct.
- 2 S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- 3 $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- 4 \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (active iteration).
- 5 $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- 6 π_{i+1} : uniform distribution over the vertices of S_i .

Proof...

- 1 Assume all distances in \mathbf{G} are distinct.
- 2 S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- 3 $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- 4 \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (active iteration).
- 5 $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- 6 π_{i+1} : uniform distribution over the vertices of S_i .

Proof...

- 1 Assume all distances in \mathbf{G} are distinct.
- 2 S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- 3 $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- 4 \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (active iteration).
- 5 $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- 6 π_{i+1} : uniform distribution over the vertices of S_i .

Proof...

- 1 Assume all distances in \mathbf{G} are distinct.
- 2 S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- 3 $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- 4 \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (**active** iteration).
- 5 $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- 6 π_{i+1} : uniform distribution over the vertices of S_i .

Proof...

- ① Assume all distances in \mathbf{G} are distinct.
- ② S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- ③ $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- ④ \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (**active** iteration).
- ⑤ $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- ⑥ π_{i+1} : uniform distribution over the vertices of S_i .

Proof...

- 1 Assume all distances in \mathbf{G} are distinct.
- 2 S_i : set of all vertices $x \in \mathbf{V}$, such that:
 - (A) $d(x, v) < d(v, \Pi_i)$, where $\Pi_i = \{\pi_1, \dots, \pi_i\}$.
 - (B) If $\pi_{i+1} = x$ then $\delta(v)$ would change in the $(i + 1)$ th iteration.
- 3 $s_i = |S_i|$. Observe $S_1 \supseteq S_2 \supseteq \dots \supseteq S_n$, and $|S_n| = 0$.
- 4 \mathcal{E}_{i+1} : event that $\delta(v)$ changed in iteration $(i + 1)$ (**active** iteration).
- 5 $(i + 1)$ iteration active: $\pi_{i+1} \in S_i$.
- 6 π_{i+1} : uniform distribution over the vertices of S_i .

Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Proof continued...

- ① \mathcal{E}_{i+1} happens then $s_{i+1} \leq s_i/2$, with probability $\geq 1/2$.
- ② iteration is **lucky**.
- ③ After $O(\log n)$ lucky iterations set S_i empty: Done.
- ④ $\mathcal{E}_1, \dots, \mathcal{E}_n$: Independent.
- ⑤ By Chernoff inequality, after $c \log n$ active iterations, at least $\lceil \log_2 n \rceil$ iterations lucky. with high probability.



Correctness continued...

Lemma

Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, with n vertices and m edges, the above algorithm computes an r -net of \mathbf{G} in $O((n + m) \log n)$ expected time.

Proof.

By above lemma, the two δ values associated with the endpoints of an edge get updated $O(\log n)$ times, in expectation, during the algorithm's execution. As such, a single edge creates $O(\log n)$ decrease-key operations in the heap maintained by the algorithm. Each such operation takes constant time if we use Fibonacci heaps to implement the algorithm. \square

