Chapter 13

Backward analysis

NEW CS 473: Theory II, Fall 2015
October 8, 2015

13.1 Some more probability

13.1.0.1 Some more probability

Lemma 13.1.1. *$\mathcal{E}_1, \ldots, \mathcal{E}_n$: n events (not necessarily independent). Then,*

$$\mathbf{Pr}\left[\cap_{i=1}^{n} \mathcal{E}_i\right] = \mathbf{Pr}\left[\mathcal{E}_1\right] * \mathbf{Pr}\left[\mathcal{E}_2 \,|\mathcal{E}_1\right] * \mathbf{Pr}\left[\mathcal{E}_3 \;\middle|\; \mathcal{E}_1 \cap \mathcal{E}_2\right] * \ldots \\ * \mathbf{Pr}\left[\mathcal{E}_n \;\middle|\; \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-1}\right].$$

13.2 Backward analysis

13.2.0.1 Backward analysis

(A) $\mathsf{P} = \langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ be a random ordering of n distinct numbers.
(B) $X_i = 1 \quad \Longleftrightarrow \quad \mathsf{p}_i$ is smaller than $\mathsf{p}_1, \ldots, \mathsf{p}_{i-1}$.
(C) **Lemma 13.2.1.** $\mathbf{Pr}[X_i = 1] = 1/i$.

13.2.0.2 Proof...

Lemma 13.2.2. $\mathbf{Pr}[X_i = 1] = 1/i$.

Proof: (A) Fix elements appearing in $\mathrm{set}(\mathsf{P}_i) = \{s_1, \ldots, s_i\}$.
(B) $\mathbf{Pr}\left[\mathsf{p}_i = \min(\mathsf{P}_i) \;\middle|\; \mathrm{set}(\mathsf{P}_i)\right] = 1/i$.

$$\begin{aligned}
&\mathbf{Pr}\left[\mathsf{p}_i = \min(\mathsf{P}_i)\right] \\
&= \sum_{S \subseteq \mathsf{P}, |S|=i} \mathbf{Pr}\left[\mathsf{p}_i = \min(\mathsf{P}_i) \;\middle|\; \mathrm{set}(\mathsf{P}_i) = S\right] \mathbf{Pr}[S] \\
&= \sum_{S \subseteq \mathsf{P}, |S|=i} \frac{1}{i} \mathbf{Pr}[S] = \frac{1}{i}.
\end{aligned}$$

13.2.1 # of times...

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

Theorem 13.2.3. *In a random permutation of n distinct numbers, the minimum of the prefix changes in expectation $\ln n + 1$ times.*

Proof: (A) $Y = \sum_{i=1}^{n} X_i$.
(B) $\mathbf{E}[Y] = \mathbf{E}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} 1/i \leq \ln n + 1$.

13.2.1.2 High probability

Lemma 13.2.4. *$\Pi = \pi_1 \dots \pi_n$: 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

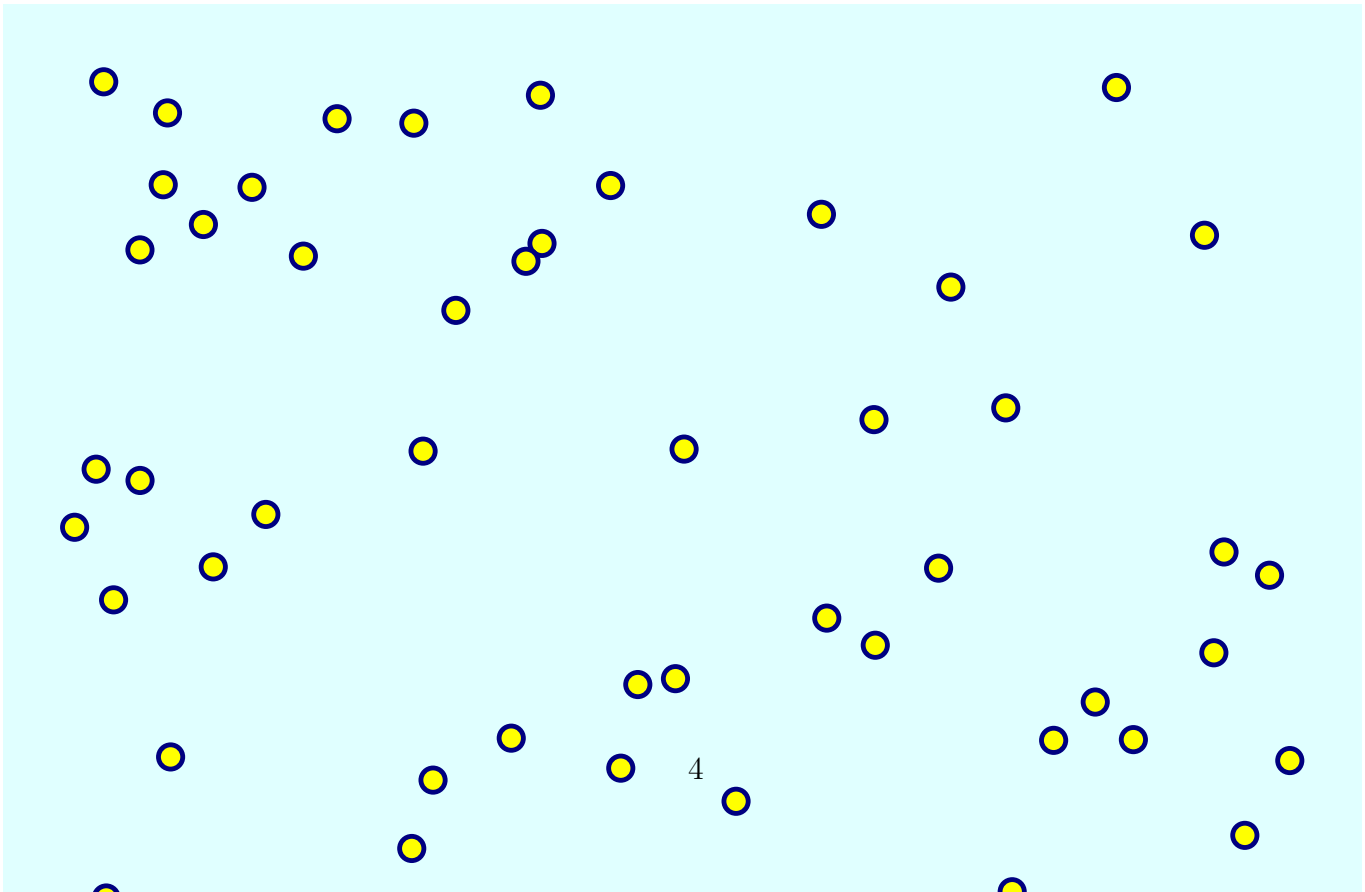
(A) $\mathcal{E}_i$: the event that $X_i = 1$, for $i = 1, \dots, n$.
(B) Claim: $\mathcal{E}_1, \dots, \mathcal{E}_n$ are independent.
(C) Generate permutation: Randomly pick a permutation of the given numbers, set first number to be π_n.
(D) Next, pick a random permutation of the remaining numbers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

13.2.1.3 Proof continued...

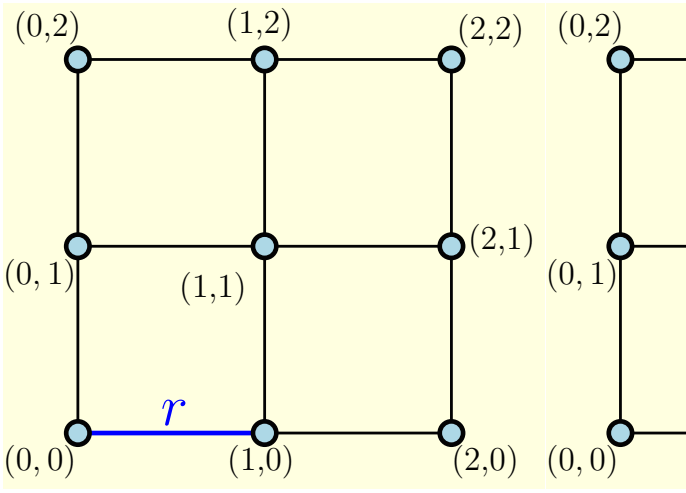
(A) For any indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, and observe that $\mathbf{Pr}\left[\mathcal{E}_{i_k} \;\middle|\; \mathcal{E}_{i_1} \cap \ldots \cap \mathcal{E}_{i_{k-1}}\right] = \mathbf{Pr}\left[\mathcal{E}_{i_k}\right]$,
(B) ..because $\mathcal{E}_{i_1}$ determined after all $\mathcal{E}_{i_2}, \dots, \mathcal{E}_k$.
(C) By induction: $\mathbf{Pr}\left[\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \cap \ldots \cap \mathcal{E}_{i_k}\right] = \mathbf{Pr}\left[\mathcal{E}_{i_1} \;\middle|\; \mathcal{E}_{i_2} \cap \ldots \cap \mathcal{E}_{i_k}\right] \mathbf{Pr}[\mathcal{E}_{i_2} \cap \ldots \cap \mathcal{E}_{i_k}] = \mathbf{Pr}\left[\mathcal{E}_{i_1}\right] \mathbf{Pr}\left[\mathcal{E}_{i_2} \cap \mathcal{E}_{i_3} \cap \ldots \cap \mathcal{E}_{i_k}\right] = \prod_{j=1}^{k} \mathbf{Pr}\left[\mathcal{E}_{i_j}\right] = \prod_{j=1}^{k} \frac{1}{i_j}$.
(D) $\implies$ variables $X_1, \dots, X_n$ are independent.
(E) Result readily follows from Chernoff's inequality. ■

13.3 Closest pair in linear time

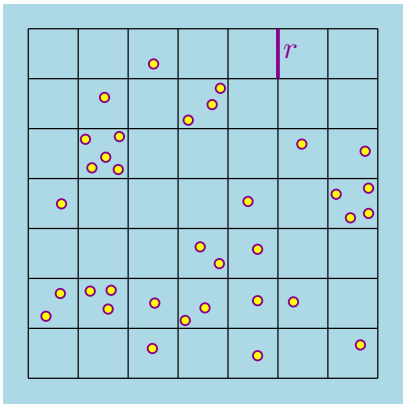
13.3.0.1 Finding the closest pair of points



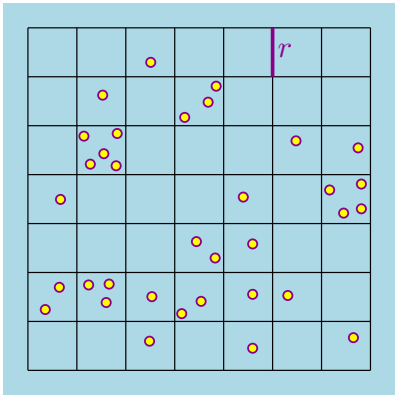
13.3.0.2 Grids...



(A) r: Side length of grid cell.
(B) Grid cell IDed by pair of integers.
(C) Constant time to determine a point p's grid cell id:
$\boxed{id(p) = (\lfloor p_x/r \rfloor, \lfloor p_y/r \rfloor)}$
(D) Limited use of the floor function (but no word packing tricks).
(E) Use hashing on (grid) points.
(F) Store points in grid...
...in linear time.



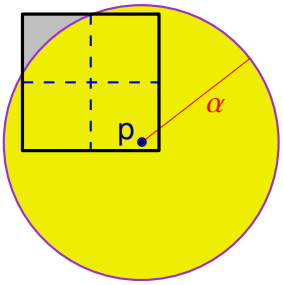
13.3.0.3 Storing point set in grid/hash-table...



Hashing:

(A) Non-empty grid cells
(B) For non-empty grid cell:
List of points in it.
(C) For a grid cell:
Its neighboring cells.

13.3.0.4 Closet pair in a square



Lemma 13.3.1. *Let* P *be a set of points contained inside a square* $\square$*, such that the sidelength of* $\square$ *is* $\alpha = \mathcal{CP}(\mathsf{P})$*. Then* $|\mathsf{P}| \leq 4$.

Proof: Partition $\square$ into four equal squares $\square_1, \ldots, \square_4$.

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

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

P can have four points if it is the four corners of $\square$. ■

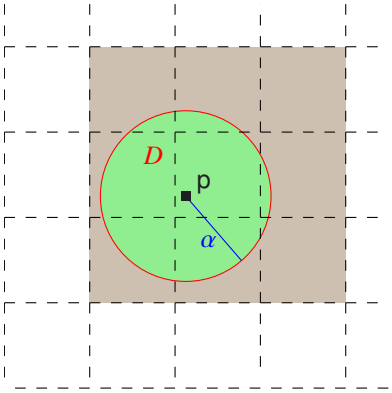
13.3.0.5 Verify closet pair

Lemma 13.3.2. P*: set of* n *points in the plane.* α*: distance. Verify in linear time whether* $\mathcal{CP}(\mathsf{P}) < \alpha$*,* $\mathcal{CP}(\mathsf{P}) = \alpha$*, or* $\mathcal{CP}(\mathsf{P}) > \alpha$*.*

proof Indeed, store the points of P in the grid 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 $\mathrm{id}(\mathsf{p})$, check if $\mathrm{id}(\mathsf{p})$ 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 $\mathrm{id}(\mathsf{p})$, just add p to it. This takes $O(n)$ time overall.

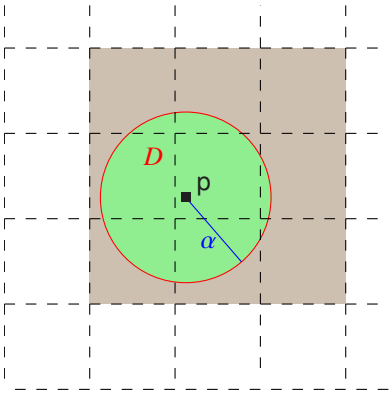
Now, if any grid cell in $\mathsf{G}_\alpha(\mathsf{P})$ contains more than, say, 4 points of P, then it must be that the $\mathcal{CP}(\mathsf{P}) < \alpha$, by previous lemma.

13.3.0.6 Proof continued



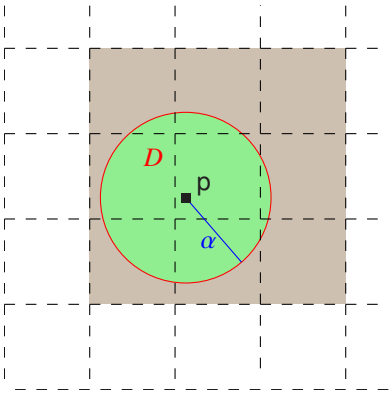
(A) When insert a point p: fetch all the points of P in cluster of P
(B) Takes constant time.
(C) If there is a point closer to p than α that was already inserted, then it must be stored in one of these 9 cells.
(D) Now, each one of those cells must contain at most 4 points of P by prev lemma.
(E) Otherwise,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

13.3.0.7 Proof continued



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

13.3.0.8 Proof continued



(A) Assume p and q: realizing closest pair.
(B) $\|\mathsf{p}-\mathsf{q}\| = \mathcal{CP}(\mathsf{P}) < \alpha$.
(C) When later point (say p) inserted, the set S would contain q.
(D) algorithm would stop and return '$\mathcal{CP}(\mathsf{P}) < \alpha$'.
(E) ■

13.3.0.9 New algorithm

(A) Pick a random permutation of the points of P.
(B) $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ be this permutation.
(C) $\alpha_2 = \|\mathsf{p}_1 - \mathsf{p}_2\|$.
(D) Insert points into the closet-pair distance verifying data-structure.
(E) α_i: the closest pair distance in the set $\mathsf{P}_i = \{\mathsf{p}_1, \ldots, \mathsf{p}_i\}$, for $i = 2, \ldots, n$.
(F) ith iteration:
  (A) if $\alpha_i = \alpha_{i-1}$. insertion takes constant time.
  (B) If $\alpha_i < \alpha_{i-1}$ then: know new closest pair distance α_i.
  (C) rebuild the grid, and reinsert the i points of P_i from scratch into the grid G_{α_i}. Takes $O(i)$ time.
(G) Returns the number α_n and points realizing it.

13.3.0.10 Weak analysis...

Lemma 13.3.3. *Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \ldots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.*

proof
(A) $X_i = 1 \iff \alpha_i < \alpha_{i-1}$.
(B) $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$ and $t = \sum_{i=3}^{n} X_i$.
(C) $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[\alpha_i < \alpha_{i-1}]$.
(D) Backward analysis. Fix P_i.
(E) $\mathsf{q} \in \mathsf{P}_i$ is ***critical*** if $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{q}\}) > \mathcal{CP}(\mathsf{P}_i)$.
(F) No critical points, then $\alpha_{i-1} = \alpha_i$ and then $\mathbf{Pr}[X_i = 1] = 0$.

13.3.0.11 Proof continued...

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

13.3.0.12 Expected linear time analysis...

Theorem 13.3.4. P*: set of* n *points in the plane. Compute the closest pair of* P *in expected linear time.*

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

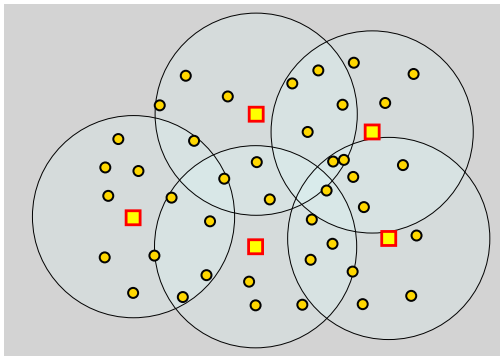
13.4 Computing nets

13.4.1 Nets

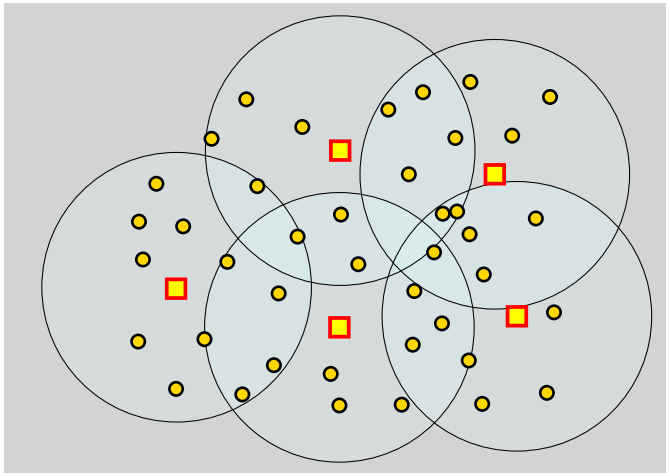
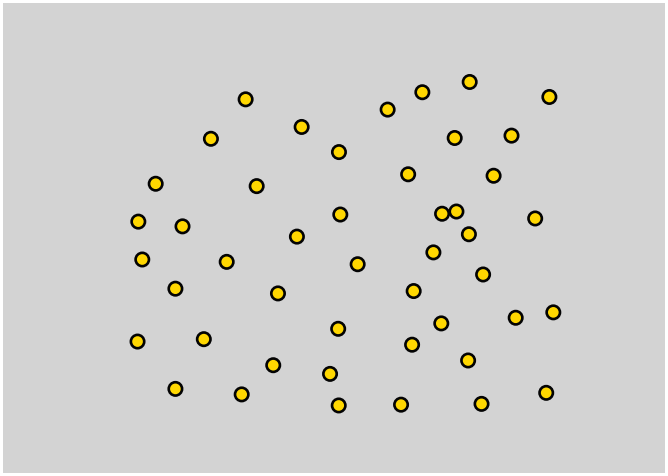
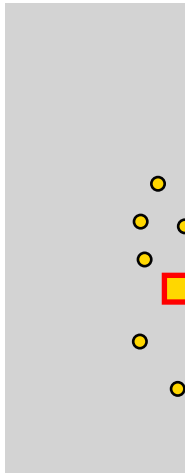
13.4.1.1 The Main Tool

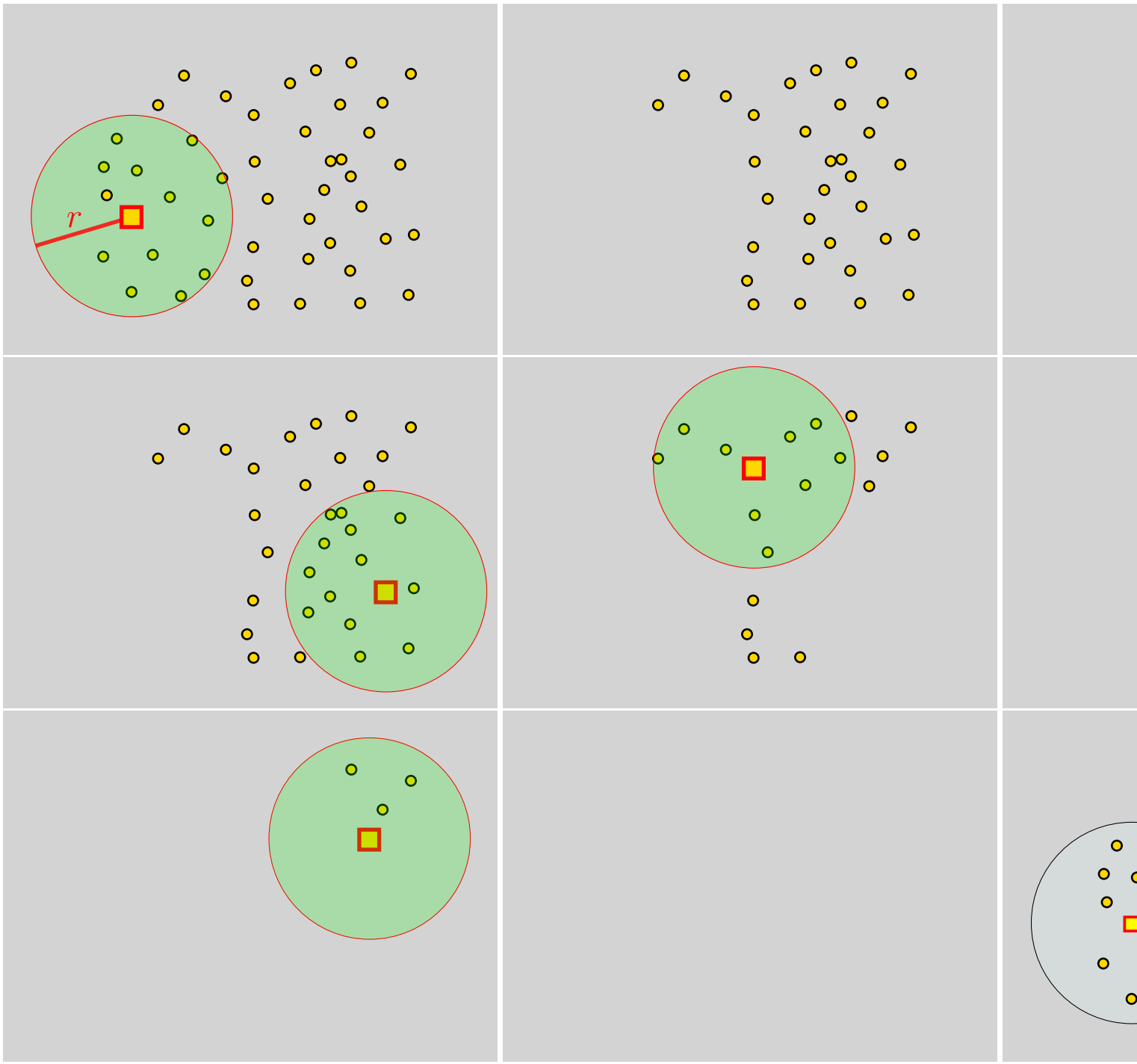
r-net $N \subseteq \mathsf{P}$ is an r-net if

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



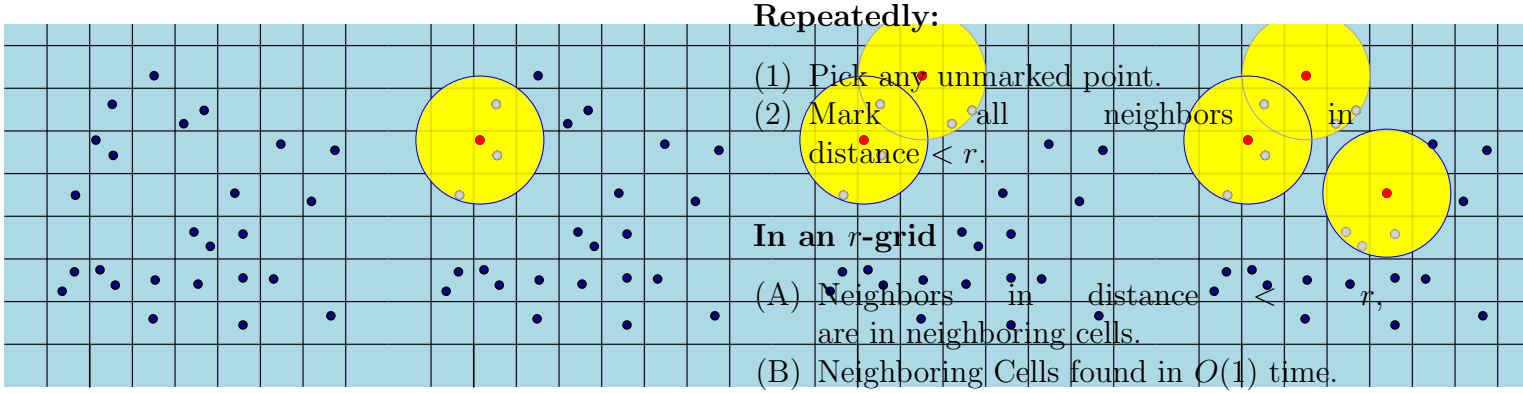
13.4.1.2 Computing an r-net


r

13.4.2 Application of Grids: Computing nets

13.4.2.1 ...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

13.5.0.1 Input

(A) $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: edge-weighted.
(B) n vertices and m edges.
(C) Task: compute an ordering $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of vertices.
(D) $\forall v \in \mathsf{V}$ L_v :
$\pi_i \in \mathrm{L}_v$, $\iff$ π_i closet vertex to v in prefix $\langle \pi_1, \ldots, \pi_i \rangle$.
(E) Example: Streaming scenario - install servers in a network.
.... every client in t network needs to know its closest server.
(F) ... client needs to maintain its current closest server.
(G) How min total size of lists? $\mathcal{L} = \sum_{v \in \mathsf{V}} |L_v|$.

13.5.0.2 Algorithm

(A) $\pi_1, \ldots, \pi_n$: random permutation of V of G.
(B) $\forall v \in \mathsf{V}$ $\delta(v) = +\infty$.
(C) For $i = 1$ to n do:
  (A) $\delta(\pi_i)$ to 0,
  (B) start Dijkstra from the ith vertex π_i.
  (C) Dijkstra propagates to u only if improves current distance.
  (D) Update $\delta(u)$ to $\mathsf{d}_{\mathsf{G}}(\pi_i, u)$ $\iff$ $\mathsf{d}_{\mathsf{G}}(\pi_i, u) < \delta'(u)$
  $\delta'(u)$: value before this iteration started.
  (E) If $\delta(u)$ updated: add π_i to L_u.

13.5.1 Analysis

13.5.1.1 Performance

Lemma 13.5.1. *Algorithm computes a permutation π, such that:*
(A) $\mathbf{E}\Big[|\mathcal{L}|\Big] = O(n \log n)$.
(B) Expected running time $O\Big((n \log n + m) \log n\Big)$.
(C) $n = |\mathsf{V}(\mathsf{G})|$ *and* $m = |\mathsf{E}(\mathsf{G})|$.

(D) Both bounds also hold with high probability.

13.5.1.2 Proof

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

13.6 Computing an r-net in a sparse graph

13.6.0.1 Computing r-net in sparse graphs.

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

13.6.0.2 Correctness

Lemma 13.6.1. *The set $\mathcal{N}$ is an r-net in G.*

Proof: (A) End: $\forall v \in \mathsf{V}$: $\delta(v) < r$.
(B) By induction: if $\ell = \delta(v)$, for some vertex v, then the distance of v to the set $\mathcal{N}$ is at most ℓ.
(C) 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 $\geq r$.

13.6.0.3 Correctness continued...

Lemma 13.6.2. *Consider an execution of the algorithm, and any vertex $v \in \mathsf{V}$. 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.*

13.6.0.4 Proof...

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

13.6.0.5 Proof continued...

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

■

13.6.0.6 Correctness continued...

Lemma 13.6.3. *Given a graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, with n vertices and m edges, the above algorithm computes an r-net of 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. ■