Local Search

Fundamental optimization problem in artificial intelligence, operations research, mathematics, engineering, physics, computational biology.

Helps model natural processes such as protein folding or insects searching for food

Local Minima on a Graph

Input. Undirected graph $G = (V, E)$ and function $f: V \to \mathbf{R}$, where $f(v)$ is the value of vertex v. Assume different values.

Local Minima on a Graph

Input. Undirected graph $G = (V, E)$ and function $f : V \to \mathbf{R}$,

where $f(v)$ is the value of vertex v. Assume different values.

Oracle access to $\boldsymbol{f}$: given a vertex v, we get $f(v)$

in one step.

Local Minima on a Graph

Input. Undirected graph $G = (V, E)$ and function $f: V \to \mathbf{R}$,

where $f(v)$ is the value of vertex v. Assume different values.

Oracle access to f: given a vertex v, we get $f(v)$

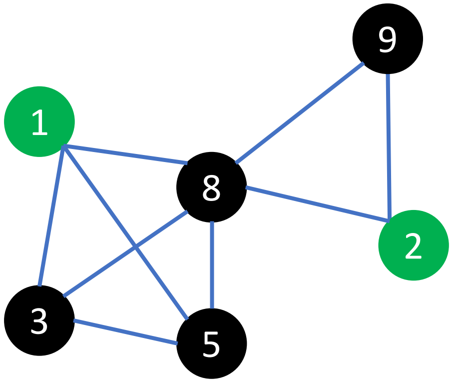
in one step.

Output. Local minimum v ∈ V: $f(v) \leq f(u)$ for all $(u, v) \in E$.

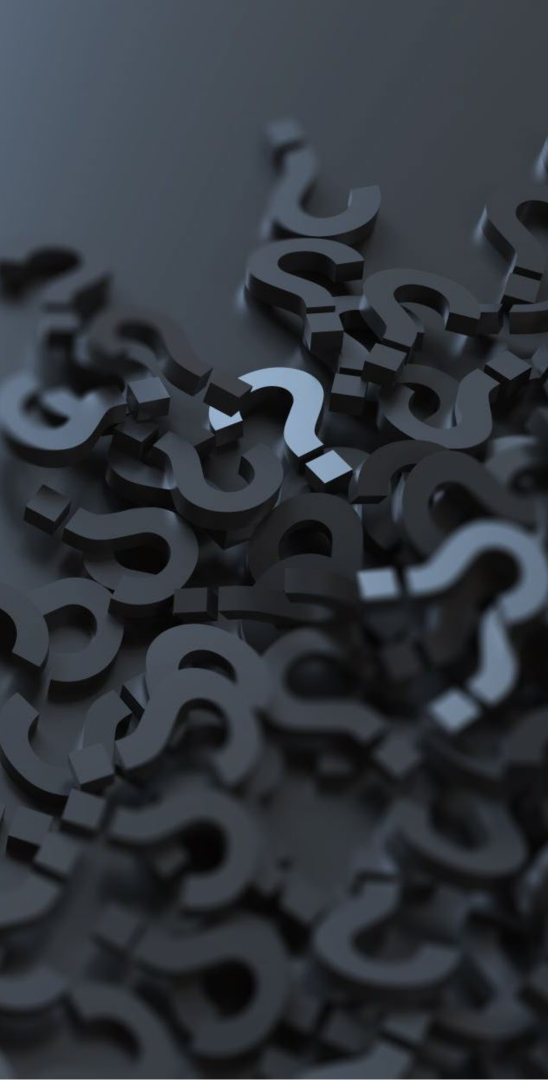
Would like to minimize the number of oracle queries.

Local Minima on a Graph

Example:



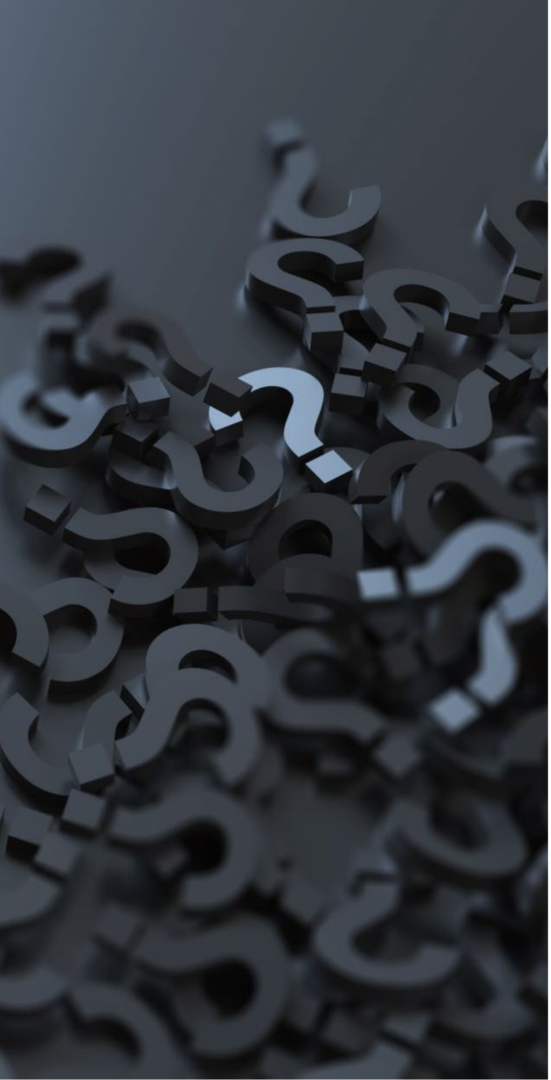
The green vertices are local minima.



Deterministic query complexity

Query complexity of a deterministic algorithm A: max # of queries made by A on an input, where the max is taken over all possible inputs.

Deterministic query complexity: the query complexity of the best deterministic algorithm.



Randomized query complexity

Query complexity of a randomized algorithm A: expected # of queries made by A on a worst case input.

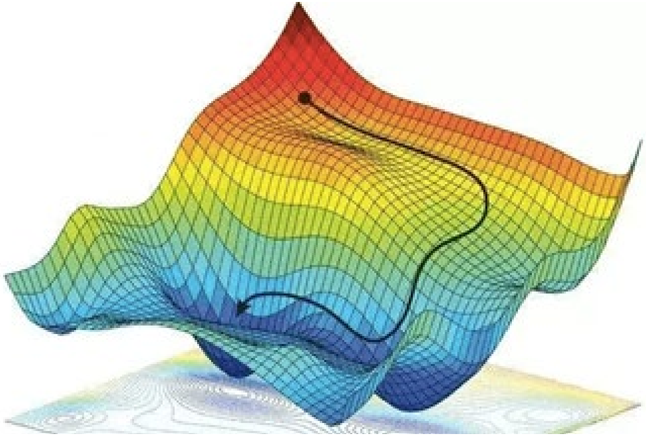
Randomized query complexity: query complexity of the best randomized algorithm that finds a solution with probability $\geq \frac{2}{3}$, where the expectation is taken over the coin tosses of the algorithm.

Naïve Steepest Descent

1. Query an arbitrary initial point x_0.
2. At each step *i*, query all the *neighbors* of x_{i-1}.

   Let x_i be the point with *minimum* value among them.

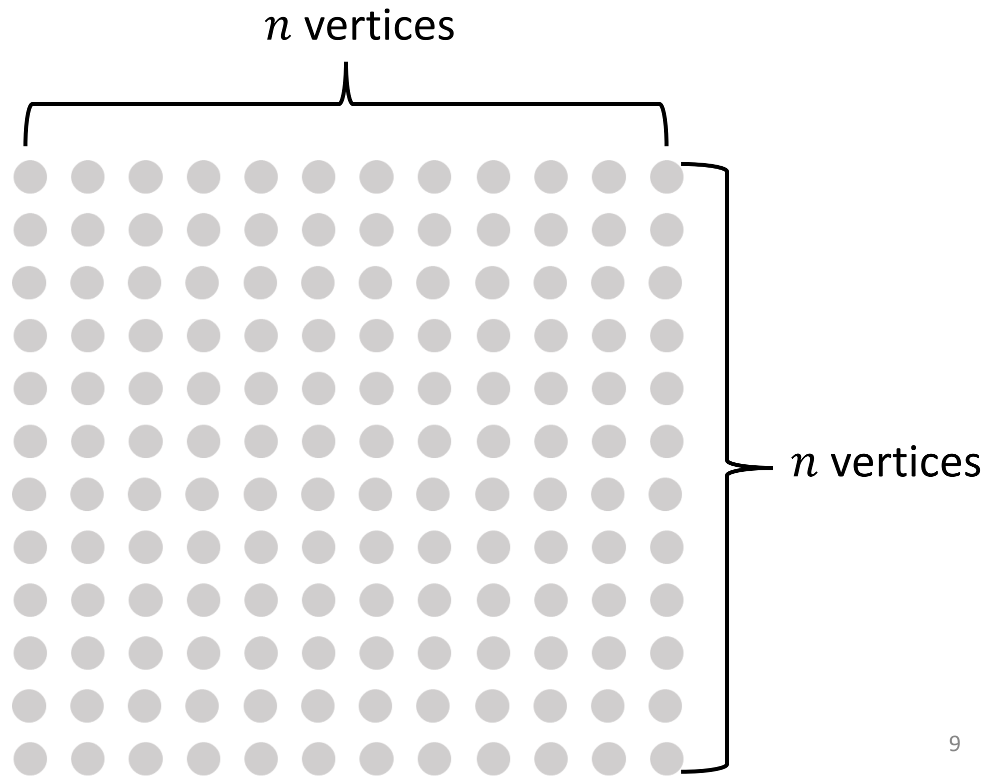
   If x_i is a local minimum then return it and stop. Otherwise continue.



This may query all the vertices in the worst case.

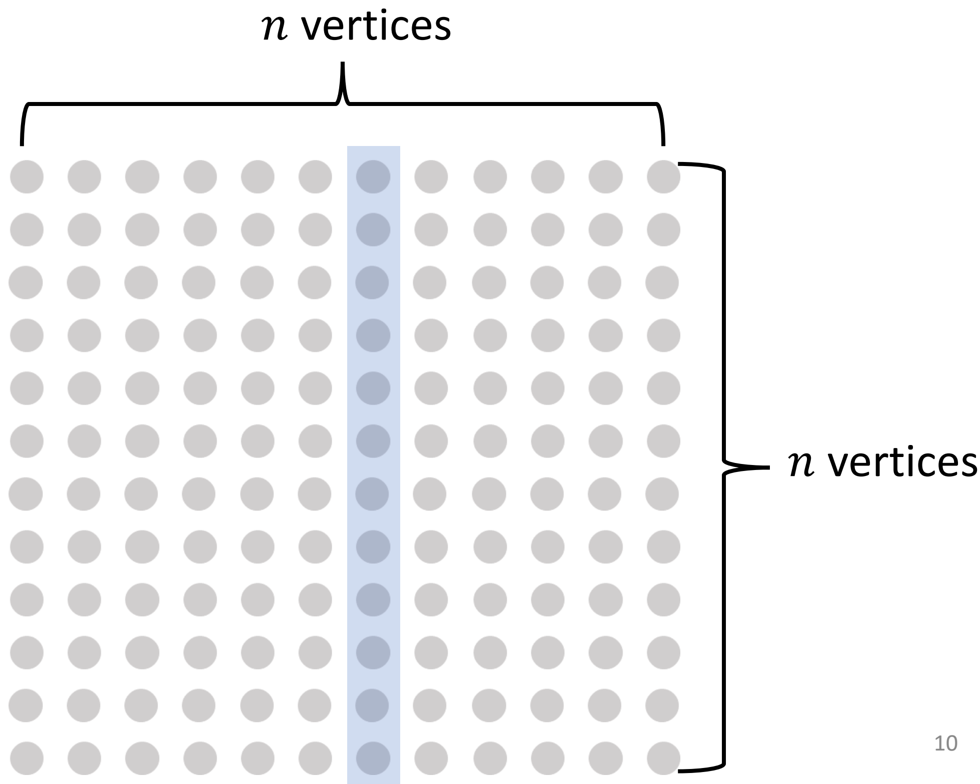
Can we do better on the 2d grid?

Challenge: design an algorithm that asks fewer than n^2 queries on the $n \times n$ grid.



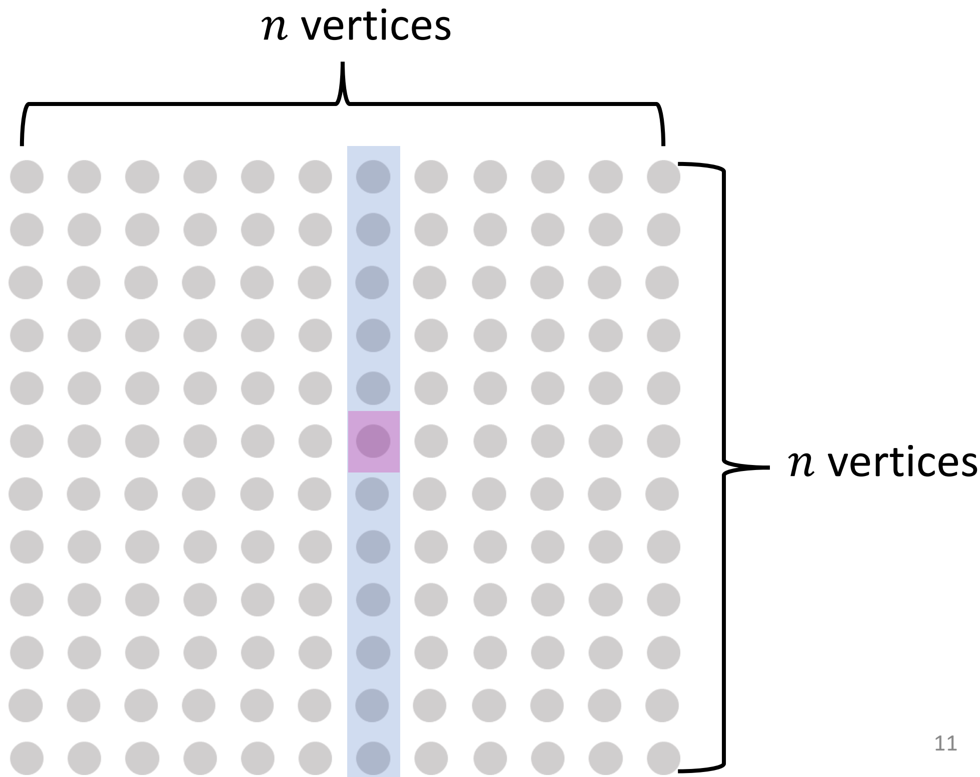
Can we do better on the 2d grid?

Query dividing line and find the minimum among those points.



Can we do better on the 2d grid?

Query dividing line and find the minimum among those points – say it's x_1.

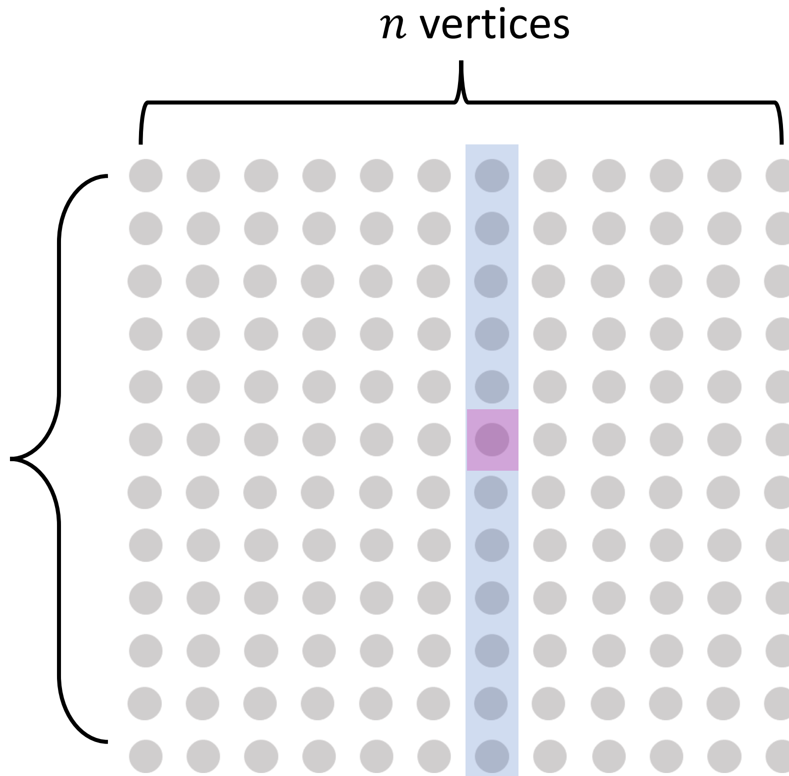


Can we do better on the 2d grid?

Query dividing line and find the minimum among those points – x_1.

Check if x_1 is is a local min.

- If yes, we are done.
- Else?



Can we do better on the 2d grid?

Query dividing line and find the minimum among those points – x_1.

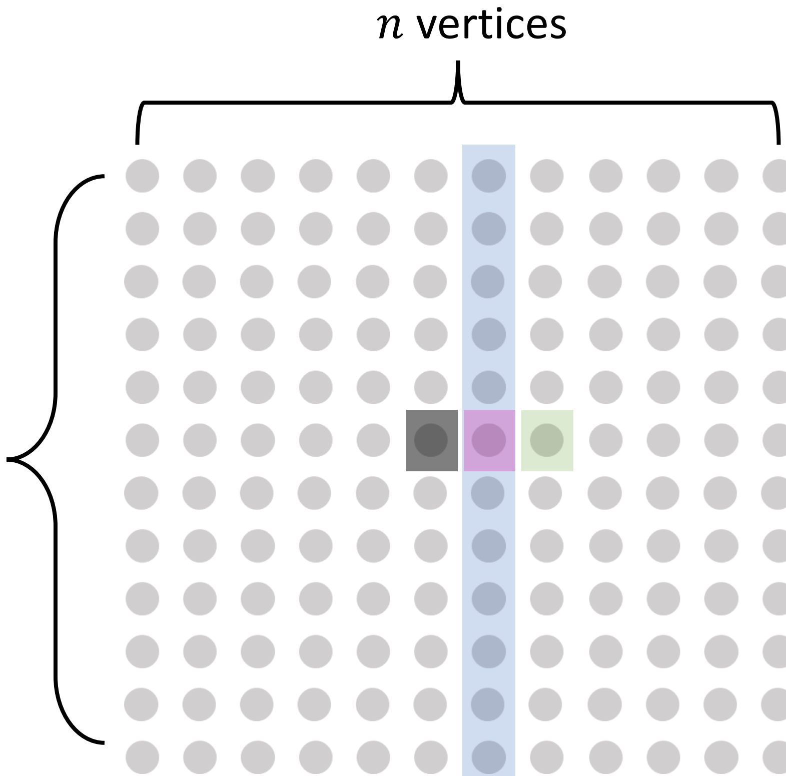
Check if x_1 is is a local min.

- If yes, we are done.
- Else recurse on the side

(left or right) containing the

smallest neighbour of x_1.

Why is this a good idea?



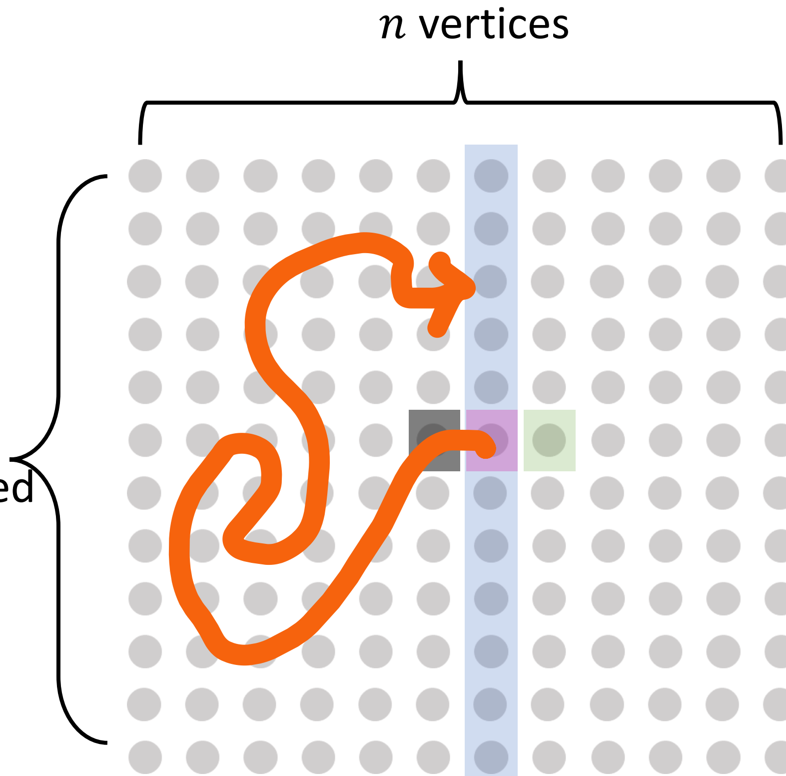
Can we do better on the 2d grid?

Query dividing line and find the minimum among those points – x_1.

Check if x_1 is is a local min.

- If yes, we are done.
- Else recurse on the side (left or right) containing the smallest neighbour of x_1.

Claim: that side is guaranteed to contain a solution.



Can we do better on the 2d grid?

Query dividing line and find the minimum among those points – x_1.

Check if x_1 is is a local min.

- If yes, we are done.
- Else recurse on the side

(left or right) containing the

smallest neighbour of x_1.

What is the number of

vertices queried in the worst

case?