CSC373

Weeks 9 & 10: Approximation Algorithms & Local Search

NP-Completeness

- We saw that many problems are NP-complete
  - ➢ Unlikely to have polynomial time algorithms to solve them
  - ➢ What can we do?

- One idea:
  - ➢ Instead of solving them exactly, solve them approximately
  - ➢ Sometimes, we might want to use an approximation algorithm even when we can compute an exact solution in polynomial time (WHY?)

Approximation Algorithms

- We'll focus on optimization problems
  - Decision problem: "Is there…where…$\geq k$?"
    - E.g. "Is there an assignment which satisfies at least k clauses of a given formula φ?"
  - Optimization problem: "Find…which maximizes…"
    - E.g. "Find an assignment which satisfies the maximum possible number of clauses from a given formula φ."
  - Recall that if the decision problem is hard, then the optimization problem is hard too

Approximation Algorithms

- There is a function $Profit$ we want to maximize or a function $Cost$ we want to minimize

- Given input instance I…
  - Our algorithm returns a solution $ALG(I)$
  - An optimal solution maximizing $Profit$ or minimizing $Cost$ is $OPT(I)$
  - Then, the approximation ratio of ALG on instance I is

$$\frac{Profit\big(OPT(I)\big)}{Profit\big(ALG(I)\big)} \quad \text{or} \quad \frac{Cost\big(ALG(I)\big)}{Cost\big(OPT(I)\big)}$$

Approximation Algorithms

- Approximation ratio of ALG on instance I is

$$\frac{Profit\big(OPT(I)\big)}{Profit\big(ALG(I)\big)} \text{ or } \frac{Cost\big(ALG(I)\big)}{Cost\big(OPT(I)\big)}$$

  - Note: These are defined to be ≥ 1 in each case.
    - 2-approximation = half the optimal profit / twice the optimal cost

- ALG has worst-case c-approximation if for each instance I…

$$Profit\big(ALG(I)\big) \geq \frac{1}{c} \cdot Profit\big(OPT(I)\big) \; or$$

$$Cost\big(ALG(I)\big) \leq c \cdot Cost\big(OPT(I)\big)$$

Note

- By default, when we say c-approximation, we will always mean c-approximation in the worst case
  - Also interesting to look at approximation in the average case when your inputs are drawn from some distribution

- Our use of approximation ratios ≥ 1 is just a convention
  - Some books and papers use approximation ratios ≤ 1 convention
  - E.g. they might say 0.5-approximation to mean that the algorithm generates at least half the optimal profit or has at most twice the optimal cost

PTAS and FPTAS

- Arbitrarily close to 1 approximations
- FPTAS: Fully polynomial time approximation scheme
  - For every $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm that runs in time $poly(n, 1/\epsilon)$ on instances of size n
- PTAS: Polynomial time approximation scheme
  - For every $\epsilon > 0$, there is a $(1+\epsilon)$-approximation algorithm that runs in time $poly(n)$ on instances of size n
    - Note: Could have exponential dependence on $1/\epsilon$

Approximation Landscape

Impossibility of better approximations assuming widely held beliefs like P $\neq$ NP

n = parameter of problem at hand

- An FPTAS
  - E.g. the knapsack problem
- A PTAS but no FPTAS
  - E.g. the makespan problem (we'll see)
- c-approximation for a constant $c > 1$ but no PTAS
  - E.g. vertex cover and JISP (we'll see)
- $\Theta(\log n)$-approximation but no constant approximation
  - E.g. set cover
- No $n^{1-\epsilon}$-approximation for any $\epsilon > 0$
  - E.g. graph coloring and maximum independent set

Makespan Minimization

Makespan

- Problem
  - Input: m identical machines, n jobs, job j requires processing time t_j
  - Output: Assign jobs to machines to minimize makespan

- Let $S[i]$ = set of jobs assigned to machine i in a solution
- Constraints:
  - Each job must run contiguously on one machine
  - Each machine can process at most one job at a time
- Load on machine i : $L_i = \sum_{j \in S[i]} t_j$
- Goal: minimize makespan $L = \max_i L_i$

Makespan

- Even the special case of $m = 2$ machines is already NP-hard by reduction from PARTITION

- PARTITION
  - Input: Set S containing n integers
  - Output: Can we partition S into two sets with equal sum (i.e. $S = S_1 \cap S_2, S_1 \cap S_2 = \emptyset$, and $\sum_{w \in S_1} w = \sum_{w \in S_2} w$)?
  - Exercise!
    - Show that PARTITION is NP-complete by reduction from SUBSET-SUM
    - Show that if there is a polynomial-time algorithm for solving MAKESPAN with 2 machines, then you can solve PARTITION in polynomial-time

Makespan

- Greedy list-scheduling algorithm
  - Consider the n jobs in some "nice" sorted order.
  - Assign each job j to a machine with the smallest load so far

- Note
  - Implementable in $O(n \log m)$ using priority queue

- Back to greedy…?
  - But this time, we can't hope that greedy will be optimal
  - We can still hope that it is approximately optimal

- Which order?

Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.
  - This was the first worst-case approximation analysis

- Let optimal makespan = L^*

- To show that makespan under greedy solution is not much worse than L^*, we need to show that L^* isn't too low

Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.
- Fact 1: $L^* \geq \max_j t_j$
  - Some machine must process job with highest processing time
- Fact 2: $L^* \geq \frac{1}{m} \sum_j t_j$
  - Total processing time is $\sum_j t_j$
  - At least one machine must do at least $1/m$ of this work (pigeonhole principle)

Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.

- Proof:
  - Suppose machine i is bottleneck under greedy (so load = L_i)
  - Let j^* = last job scheduled on i by greedy
  - Right before j^* was assigned to i, i had the smallest load
    - Load of other machines could have only increased from then
    - $L_i - t_{j^*} \leq L_k, \forall k$
  - Average over all k : $L_i - t_{j^*} \leq \frac{1}{m} \sum_j t_j$
  - $L_i \leq t_{j^*} + \frac{1}{m} \sum_j t_j \leq L^* + L^* = 2L^*$

Fact 1

Fact 2

Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.

- Is our analysis tight?
  - Essentially.
  - There is an example where greedy does perform this badly.

  - Note: In the upcoming example, greedy is only as bad as $2 - 1/m$, but you can also improve earlier analysis to show that greedy always gives $2 - 1/m$ approximation.
  - So $2 - 1/m$ is exactly tight.

Makespan

- Theorem [Graham 1966]
  - Regardless of the order, greedy gives a 2-approximation.

- Is our analysis tight?
  - Example:
    - $m(m-1)$ jobs of length 1, followed by one job of length m
    - Greedy evenly distributes unit length jobs on all m machines, and assigning the last heavy job makes makespan $m-1+m=2m-1$
    - Optimal makespan is m by evenly distributing unit length jobs among $m-1$ machines and putting the single heavy job on the remaining

  - Idea: It seems keeping heavy jobs at the end is bad. So just start with them first!

Makespan

- Longest Processing Time (LPT) First
  - Run the greedy algorithm but consider jobs in the decreasing order of their processing time
- Need more facts about what the optimal cannot beat
- Fact 3: If the bottleneck machine has only one job, then the solution is optimal.
  - The optimal solution must schedule that job on some machine

Makespan

- Longest Processing Time (LPT) First
  - Run the greedy algorithm but consider jobs in the decreasing order of their processing time
  - Suppose $t_1 \geq t_2 \geq \cdots \geq t_n$
- Fact 4: If there are more than m jobs, $L^* \geq 2 \cdot t_{m+1}$
  - Consider the first $m + 1$ jobs
  - All of them require processing time at least t_{m+1}
  - By pigeonhole principle, in the optimal solution, at least two of them end up on the same machine

Makespan

- Theorem
  - Greedy with longest processing time first gives 3/2-approximation
- Proof:
  - Similar to the proof for arbitrary ordering
  - Consider bottleneck machine i and job j^* that was last scheduled on this machine by greedy
  - Case 1: Machine i has only one job j^*
    - By Fact 3, greedy is optimal in this case (i.e. 1-approximation)

Makespan

- Theorem
  - Greedy with longest processing time first gives 3/2-approximation
- Proof:
  - Similar to the proof for arbitrary ordering
  - Consider bottleneck machine i and job j^* that was last scheduled on this machine by greedy
  - Case 2: Machine i has at least two jobs
    - Job j^* must have $t_{j^*} \leq t_{m+1}$
    - As before, $L = L_i = \underbrace{\left(L_i - t_{j^*}\right)}_{\leq L^*} + \underbrace{t_{j^*}}_{\leq L^*/2} \leq 1.5\, L^*$

Same as before

$t_{j^*} \leq t_{m+1}$ and Fact 4

Makespan

- Theorem
  - Greedy with LPT rule gives $3/2$-approximation
  - Is our analysis tight? No!
- Theorem [Graham 1966]
  - Greedy with LPT rule gives $4/3$-approximation
  - Is Graham's 4/3 approximation tight?
    - Essentially.
    - In the upcoming example, greedy is only as bad as $\frac{4}{3} - \frac{1}{3m}$
    - But Graham actually proves $\frac{4}{3} - \frac{1}{3m}$ upper bound. So this is exactly tight.

Makespan

- Theorem
  - Greedy with LPT rule gives $3/2$-approximation
  - Is our analysis tight? No!

- Theorem [Graham 1966]
  - Greedy with LPT rule gives $4/3$-approximation
  - Tight example:
    - 2 jobs of lengths $m, m+1, \dots, 2m-1$, one more job of length m
    - Greedy-LPT has makespan $4m-1$ (verify!)
    - OPT has makespan $3m$ (verify!)
    - Thus, approximation ratio is at least as bad as $\frac{4m-1}{3m} = \frac{4}{3} - \frac{1}{3m}$

Unweighted Vertex Cover

Unweighted Vertex Cover

- Problem
  - Input: Undirected graph $G = (V, E)$
  - Output: Vertex cover S of minimum cardinality

  - Recall: S is vertex cover if every edge has at least one endpoint in S
  - We already saw that this problem is NP-hard

- Q: What would be a good greedy algorithm for this problem?

Unweighted Vertex Cover

- Greedy edge-selection algorithm:
  - Start with $S = \emptyset$
  - While there exists an edge whose both endpoints are not in S, add *both* its endpoints to S

- Hmm…
  - Why are we selecting edges rather than vertices?
  - Why are we adding both endpoints?
  - We'll see..

Unweighted Vertex Cover

GREEDY-VERTEX-COVER(G)

$S \leftarrow \varnothing$.

$E' \leftarrow E$.

WHILE ($E' \neq \varnothing$)

Let $(u, v) \in E'$ be an arbitrary edge.

$M \leftarrow M \cup \{(u, v)\}$. ← *M* is a matching

$S \leftarrow S \cup \{u\} \cup \{v\}$. ← every vertex cover must take at least one of these; we take both

Delete from E' all edges incident to either u or v.

RETURN S.

Unweighted Vertex Cover

- Theorem:
  - Greedy edge-selection algorithm for unweighted vertex cover gives 2-approximation.

- Question:
  - If S is any vertex cover (containing $|S|$ vertices), M is any matching (containing $|M|$ edges), then what is the relation between $|S|$ and $|M|$?
  - Answer: $|S| \geq |M|$.

Unweighted Vertex Cover

- Theorem:
  - Greedy edge-selection algorithm for unweighted vertex cover gives 2-approximation.
- Proof:
  - Let S^* = min vertex cover, S = solution returned by greedy
  - By design, $|S| = 2 \cdot |M|$
  - Because M is a matching, $|S^*| \geq |M|$ (By last slide)
  - Hence, $|S| \leq 2|S^*|$ ■

Unweighted Vertex Cover

- Theorem:
  - Greedy edge-selection algorithm for unweighted vertex cover gives 2-approximation.
- Corollary:
  - If M^* is maximum matching, then greedy finds matching M with $|M| \geq \frac{1}{2}|M^*|$
- Proof:
  - By design, $|M| = \frac{1}{2}|S|$
  - $|S| \geq |M^*|$ (Same reason again!)
  - Hence, $|M| \geq \frac{1}{2}|M^*|$ ■

This is a so-called *maximal* matching which cannot be *extended*

Unweighted Vertex Cover

- What about a greedy vertex selection algorithm?
  - Start with $S = \emptyset$
  - While S is not a vertex cover:
    - Choose a vertex v which maximizes the number of uncovered edges incident on it
    - Add v to S
  - Interestingly, this only gives $\log d_{\max}$ approximation, where $d_{\max}$ is the maximum degree of any vertex
    - But unlike the edge-selection version, this generalizes to set cover, and gives provably best possible approximation ratio for set cover in polynomial time (unless P=NP)

Unweighted Vertex Cover

- Theorem [Dinur-Safra 2004]:
  - Unless P = NP, there is no ρ-approximation polynomial-time algorithm for unweighted vertex cover for any $\rho < 1.3606$.

On the Hardness of Approximating Minimum Vertex Cover

Irit Dinur* Samuel Safra†

May 26, 2004

Abstract

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.

Unweighted Vertex Cover

- Theorem [Dinur-Safra 2004]:
  - Unless P = NP, there is no ρ-approximation polynomial-time algorithm for unweighted vertex cover for any $\rho < 1.3606$.

- Q: How can something like this be proven?
  - We'll see later.
  - Basically, reduce "solving a hard problem" (e.g. 3SAT) to "finding any good approximation of current problem"

Weighted Vertex Cover

Weighted Vertex Cover

- Problem
  - Input: Undirected graph $G = (V, E)$, weights $w : V \to R_{\geq 0}$
  - Output: Vertex cover S of minimum total weight

- The same greedy algorithm doesn’t work
  - Gives arbitrarily bad approximation
  - Obvious modification which try to take weights into account also don’t work
  - Need another strategy…

ILP Formulation

- For each vertex v, create a binary variable $x_v \in \{0,1\}$ indicating whether vertex v is chosen in the vertex cover
- Then, computing min weight vertex cover is equivalent to solving the following integer linear program

$$\min \Sigma_v \, w_v \cdot x_v$$

subject to

$$x_u + x_v \geq 1, \qquad \forall (u,v) \in E$$

$$x_v \in \{0,1\}, \qquad \forall v \in V$$

LP Relaxation

- What if we solve this LP instead of the original ILP?

ILP with binary variables		**LP with real variables**	
$\min \Sigma_v \, w_v \cdot x_v$		$\min \Sigma_v \, w_v \cdot x_v$	
subject to		subject to	
$x_u + x_v \geq 1,$	$\forall (u, v) \in E$	$x_u + x_v \geq 1,$	$\forall (u, v) \in E$
$x_v \in \{0,1\},$	$\forall v \in V$	$x_v \geq 0,$	$\forall v \in V$

Rounding LP Solution

- What if we solve this LP instead of the original ILP?
  - Minimizes objective over a larger feasible space
  - Optimal LP objective value $\leq$ optimal ILP objective value
  - But optimal LP solution x^* is not a binary vector
    - Can we round it to some binary vector $\hat{x}$ without increasing the objective value too much?

ILP with binary variables

$$\min \Sigma_v \, w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u,v) \in E$$
$$x_v \in \{0,1\}, \qquad \forall v \in V$$

LP with real variables

$$\min \Sigma_v \, w_v \cdot x_v$$
subject to
$$x_u + x_v \geq 1, \qquad \forall (u,v) \in E$$
$$x_v \geq 0, \qquad \forall v \in V$$

Rounding LP Solution

- Consider LP optimal solution x^*
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise
  - Claim 1: $\hat{x}$ is a feasible solution of ILP (i.e. a vertex cover)
    - For every edge $(u, v) \in E$, at least one of $\{x_u^*, x_v^*\}$ is at least 0.5
    - So at least one of $\{\hat{x}_u, \hat{x}_v\}$ is 1 ■

ILP with binary variables

$$\begin{aligned} &\min \textstyle\sum_v w_v \cdot x_v \\ &\text{subject to} \\ &x_u + x_v \geq 1, \qquad \forall (u, v) \in E \\ &x_v \in \{0,1\}, \qquad \forall v \in V \end{aligned}$$

LP with real variables

$$\begin{aligned} &\min \textstyle\sum_v w_v \cdot x_v \\ &\text{subject to} \\ &x_u + x_v \geq 1, \qquad \forall (u, v) \in E \\ &x_v \geq 0, \qquad \forall v \in V \end{aligned}$$

Rounding LP Solution

- Consider LP optimal solution x^*
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise
  - Claim 2: $\sum_v w_v \cdot \hat{x}_v \leq 2 * \sum_v w_v \cdot x_v^*$
    - Weight only increases when some $x_v^* \in [0.5,1]$ is shifted *up* to 1
    - At most doubling the variable, so at least doubling the weight ■

ILP with binary variables

$\min \Sigma_v w_v \cdot x_v$
subject to
$x_u + x_v \geq 1, \quad \forall (u,v) \in E$
$x_v \in \{0,1\}, \quad \forall v \in V$

LP with real variables

$\min \Sigma_v w_v \cdot x_v$
subject to
$x_u + x_v \geq 1, \quad \forall (u,v) \in E$
$x_v \geq 0, \quad \forall v \in V$

Rounding LP Solution

- Consider LP optimal solution x^*
  - Let $\hat{x}_v = 1$ whenever $x_v^* \geq 0.5$ and $\hat{x}_v = 0$ otherwise
  - Hence, $\hat{x}$ is a vertex cover with weight at most $2 * \text{LP}$ optimal value $\leq 2 * \text{ILP}$ optimal value

ILP with binary variables

$\min \Sigma_v w_v \cdot x_v$
subject to
$x_u + x_v \geq 1, \quad \forall (u,v) \in E$
$x_v \in \{0,1\}, \quad \forall v \in V$

LP with real variables

$\min \Sigma_v w_v \cdot x_v$
subject to
$x_u + x_v \geq 1, \quad \forall (u,v) \in E$
$x_v \geq 0, \quad \forall v \in V$

General LP Relaxation Strategy

- Your NP-complete problem amounts to solving
  - Max $c^T x$ subject to $Ax \leq b, x \in \mathbb{N}$ (need not be binary)

- Instead, solve:
  - Max $c^T x$ subject to $Ax \leq b, x \in \mathbb{R}_{\geq 0}$ (LP relaxation)
    - LP optimal value $\geq$ ILP optimal value (for maximization)
  - x^* = LP optimal solution
  - Round x^* to $\hat{x}$ such that $c^T \hat{x} \geq \frac{c^T x^*}{\rho} \geq \frac{\text{ILP optimal value}}{\rho}$
  - Gives ρ-approximation
    - Info: Best ρ you can hope to get via this approach for a particular LP-ILP combination is called the *integrality gap*

k-Center Problem

k-Center Problem

- Problem
  - Input: Set of n sites $s_1, \dots, s_n$ and an integer k
  - Output: Return a set C of k centers s.t. the maximum distance of any site from its nearest center is minimized
    - Minimize $r(C) = \max_{i \in \{1,\dots,n\}} d(s_i, C)$, where $d(s_i, C) = \min_{c \in C} d(s_i, c)$

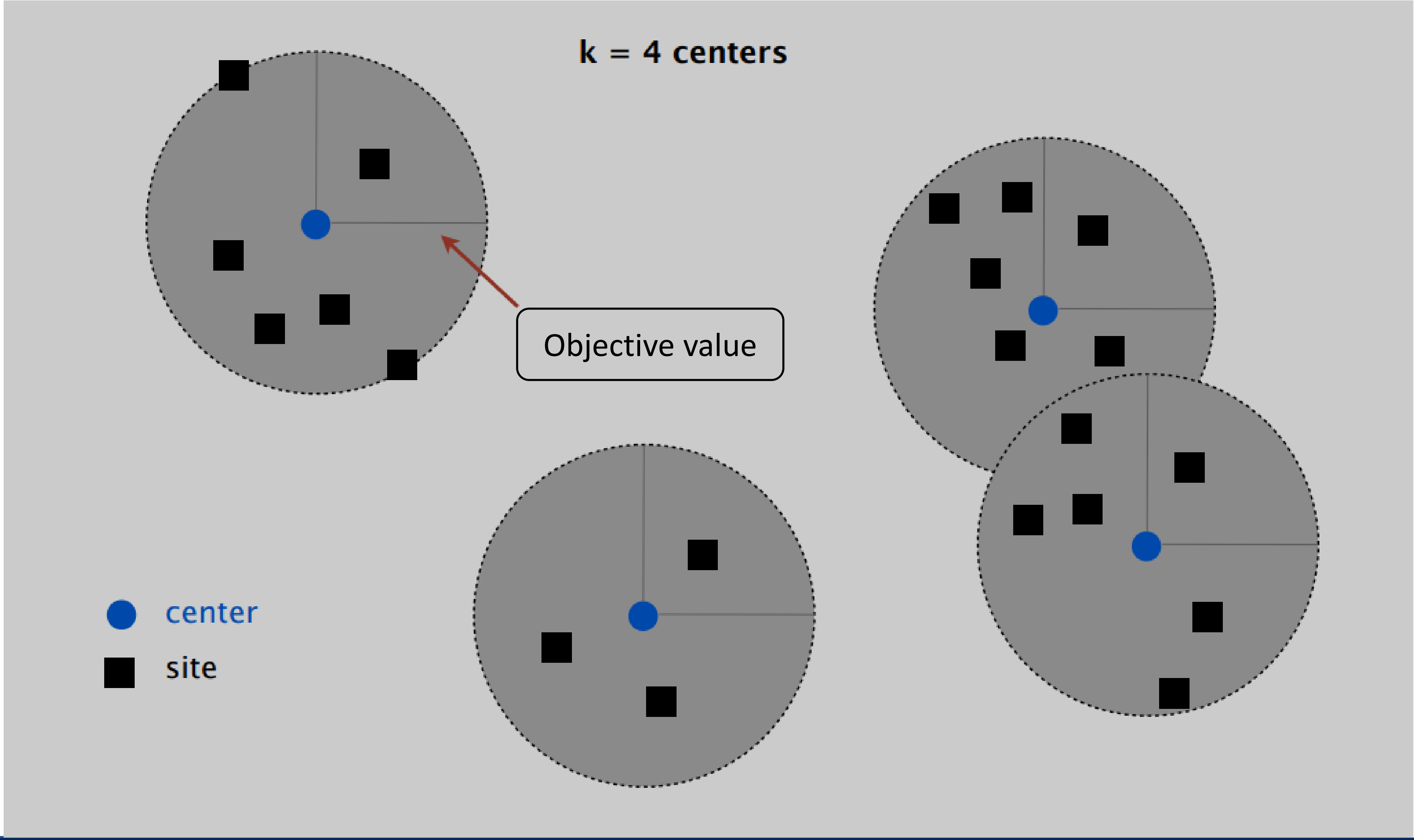
- Sites are points in some metric space with distance d satisfying:
  - Identity: $d(x, x) = 0$ for all x
  - Symmetry: $d(x, y) = d(y, x)$ for all x, y
  - Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z

k-Center Problem

- Problem
  - Input: Set of n sites $s_1, \dots, s_n$ and an integer k
  - Output: Return a set C of k centers s.t. the maximum distance of any site from its nearest center is minimized
    - Minimize $r(C) = \max_{i \in \{1,\dots,n\}} d(s_i, C)$, where $d(s_i, C) = \min_{c \in C} d(s_i, c)$

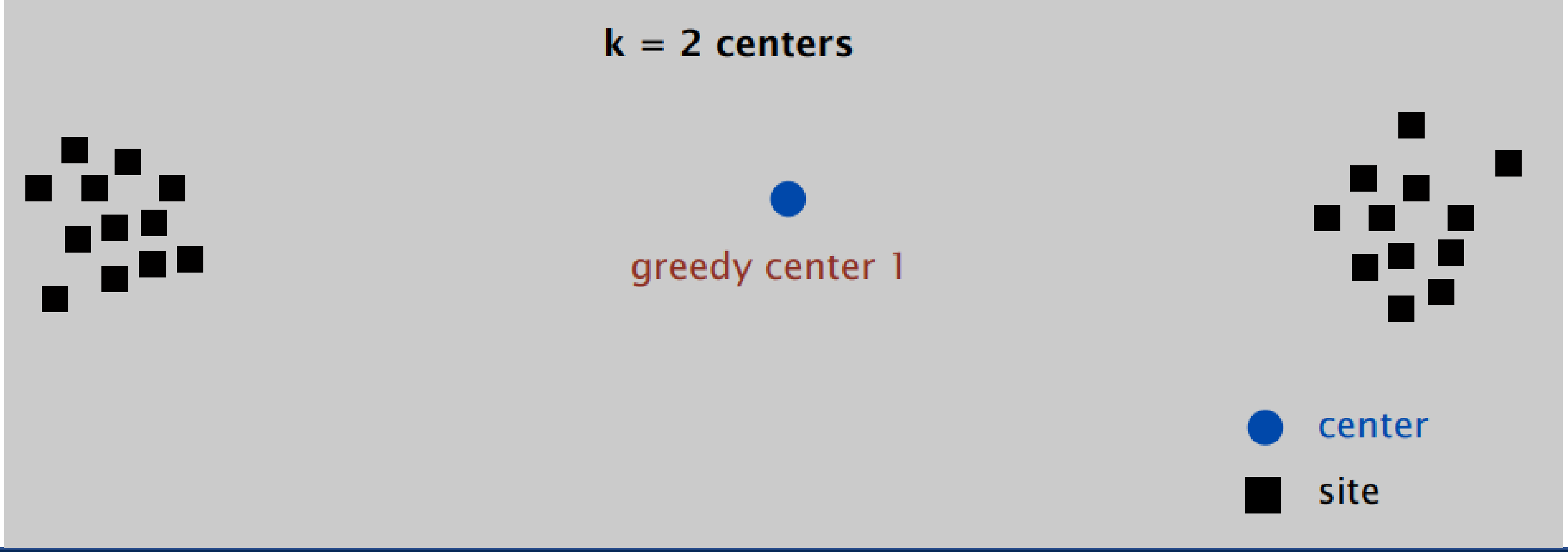
- Given C, note that $r(C)$ is the minimum radius r such that if we draw a ball of radius r around every center in C, then the balls collectively cover all the sites

k-Center Problem



Bad Greedy

- Bad greedy (forget about running time)
  - Put the first center at the optimal location for $k = 1$
  - Put every next center to reduce the objective value as much as possible given the centers already placed
- Arbitrarily bad approximation



Good Greedy

- Good greedy
  - Put the first center at an arbitrary site
  - Put every next center at a site whose distance to its nearest center is maximum among all sites

❑ Good Greedy

- $C_1 \leftarrow s_1$ (arbitrary site works)
- For $j = 2, \dots, k$:
  - $s_i \leftarrow \underset{s}{\operatorname{argmax}}\, d(s, C_{j-1})$; $\Delta_j = d\left(s_i, C_{j-1}\right)$
  - $C_{\mathrm{j}} \leftarrow C_{\mathrm{j}-1} \cup \{s_i\}$
- Return C_{k}

Good Greedy

- Good Greedy
  - $C_1 \leftarrow s_1$ (arbitrary site works)
  - For $j = 2, \dots, k$:
    - $s_i \leftarrow \underset{s}{\operatorname{argmax}}\, d(s, C_{j-1})$; $\Delta_j = d\left(s_i, C_{j-1}\right)$
    - $C_{\mathrm{j}} \leftarrow C_{\mathrm{j}-1} \cup \{s_i\}$
  - Return C_{k}

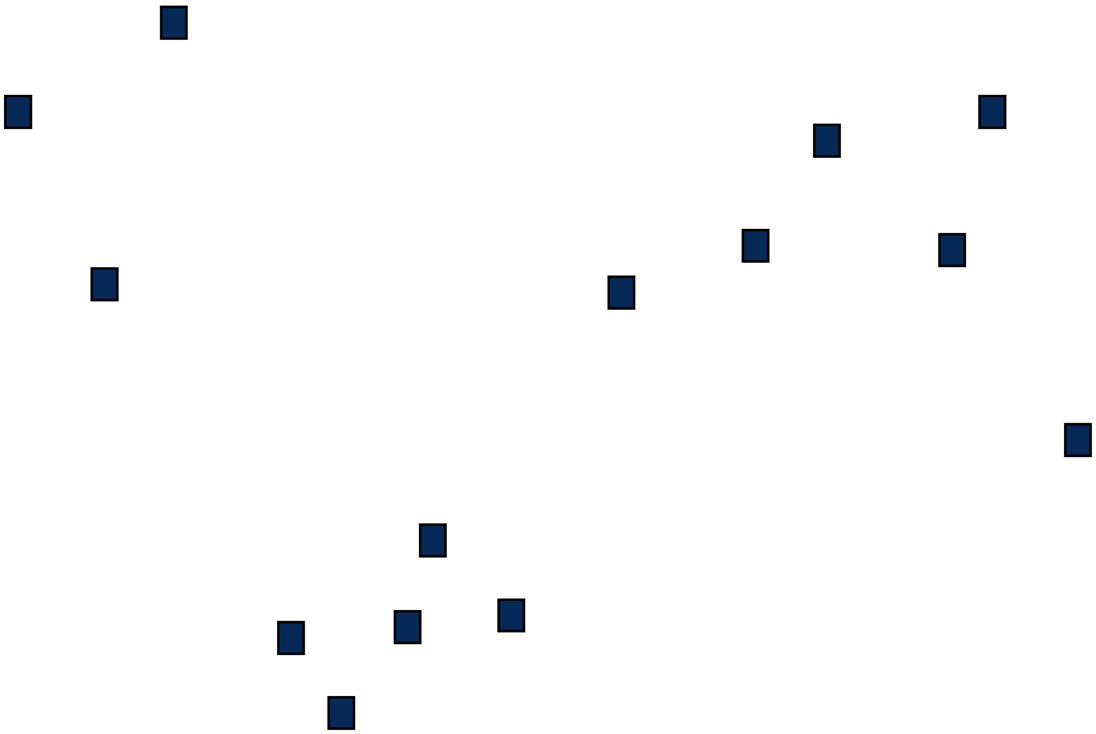
- For reasons that will soon become clear…
  - Imagine that we run good greedy for $k+1$ steps rather than k steps, and obtain C_{k+1}
  - Note: The $k+1$ points in C_{k+1} are sites

Good Greedy

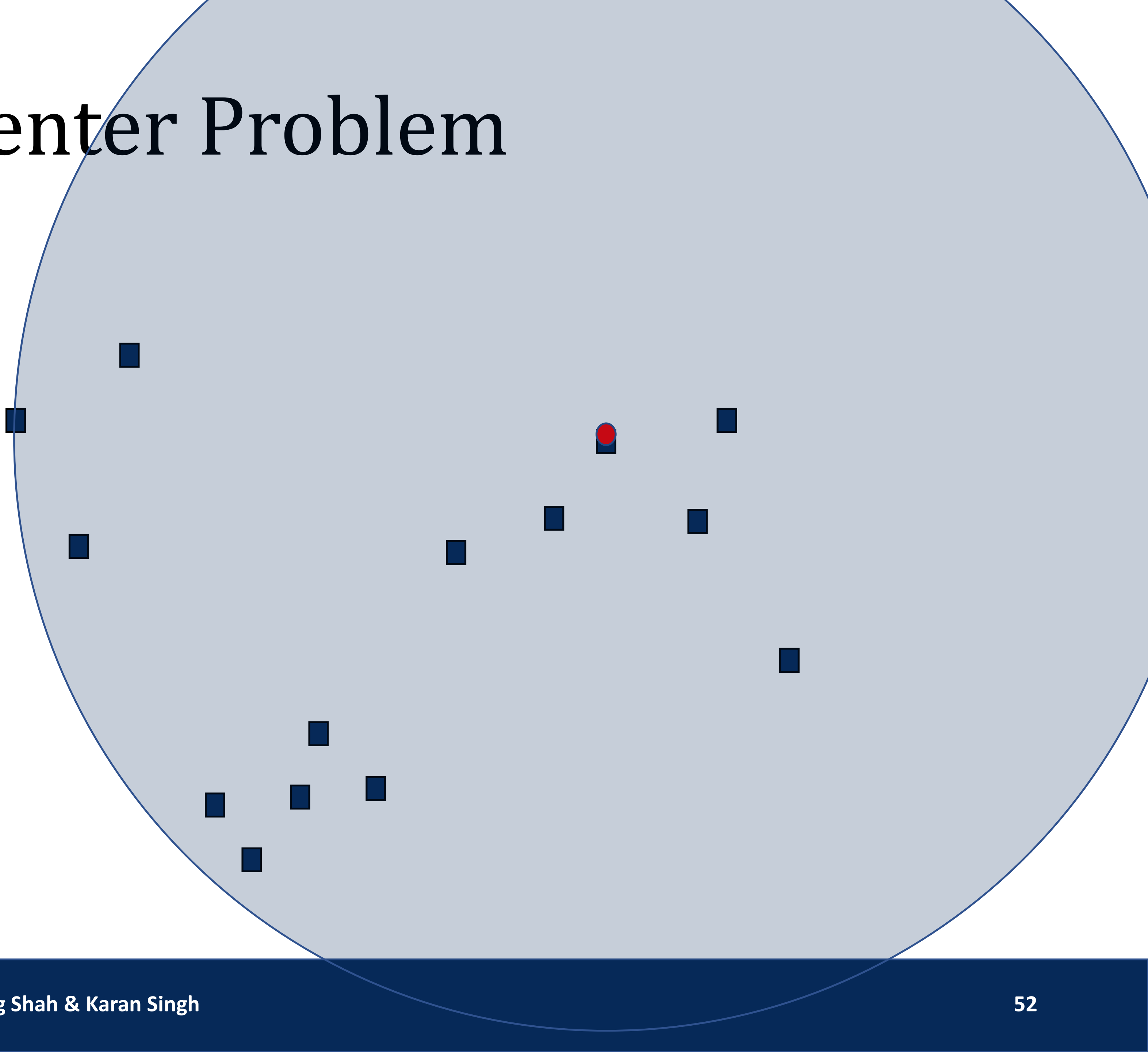
- Good Greedy
  - $C_1 \leftarrow s_1$ (arbitrary site works)
  - For $j = 2, \dots, k$:
    - $s_i \leftarrow \underset{s}{\operatorname{argmax}}\, d(s, C_{j-1})$; $\Delta_j = d\left(s_i, C_{j-1}\right)$
    - $C_\mathrm{j} \leftarrow C_{\mathrm{j}-1} \cup \{s_i\}$
  - Return C_k

- Claim: $d\left(s_i, s_j\right) \geq r(C_k)$ for all $s_i, s_j \in C_{k+1}$
  - Proof: By construction of the algorithm.
    - At each iteration j, we add a new center that is at least Δ_j far from all previous centers
    - Δ_j decreases as j increases (Why?)
    - $\Delta_{k+1} = r(C_k)$

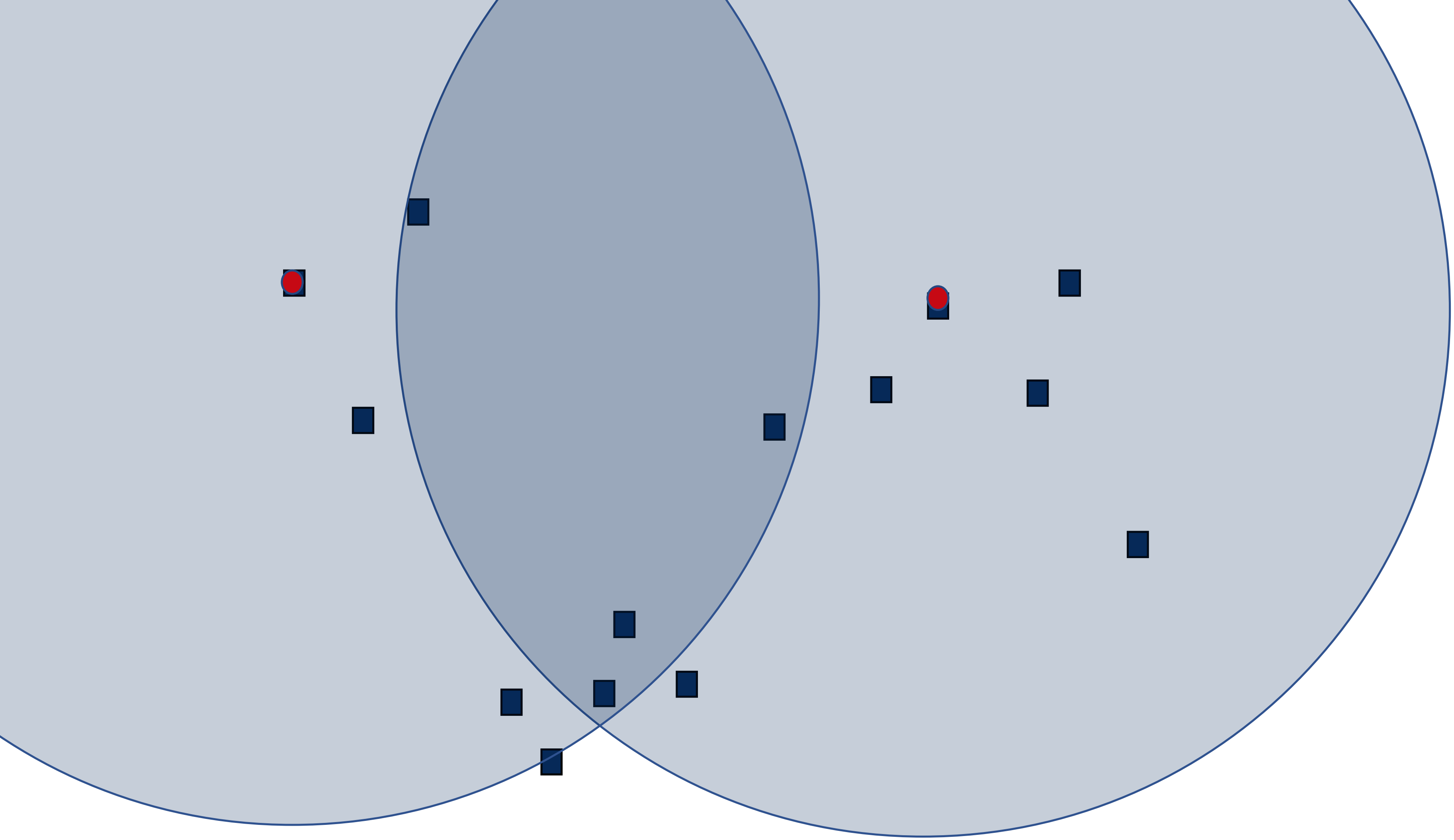
k-Center Problem



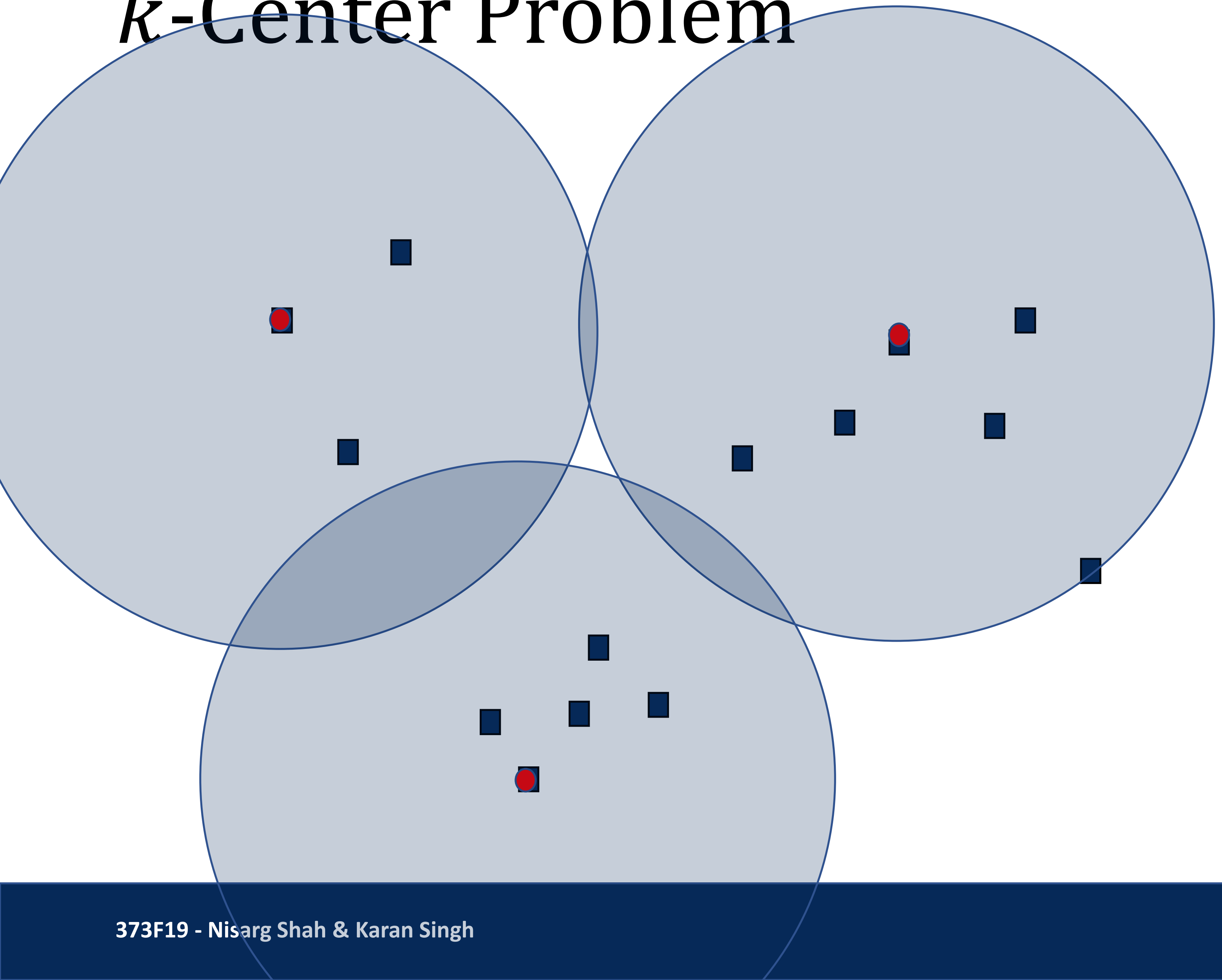
k-Center Problem



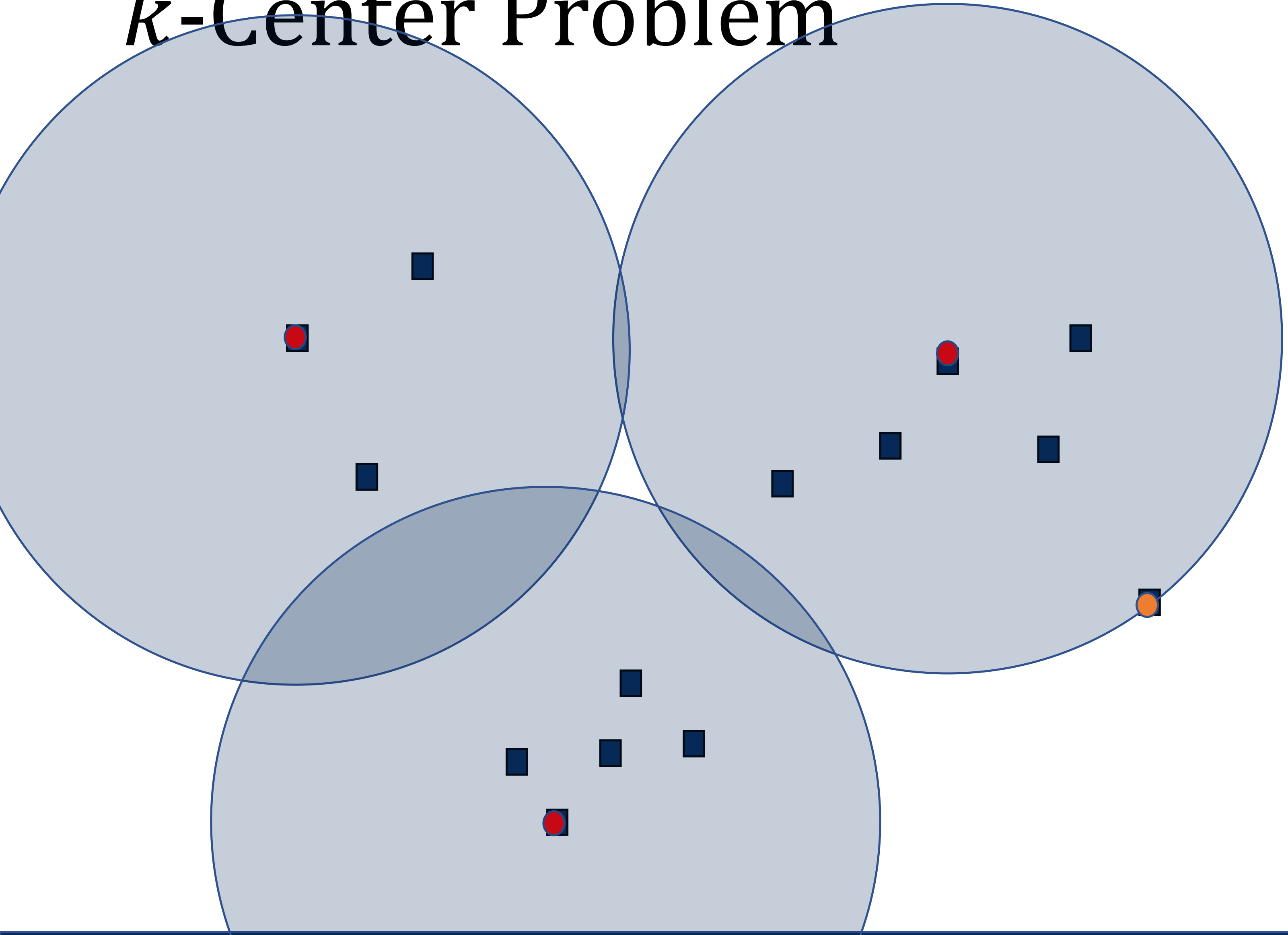
k-Center Problem



k-Center Problem



k-Center Problem



Good Greedy

- Theorem: If C^* is the optimal set of k centers, then $r(C_k) \leq 2 \cdot r(C^*)$
- Proof:
  - Draw a ball of radius $r(C^*)$ from each center in C^*
  - By pigeonhole principle, at least two $s_i, s_j \in C_{k+1}$ must belong to the same ball (say centered at $c^* \in C^*$)
    - Hence, $d(s_i, c^*), d(s_j, c^*) \leq r(C^*)$
  - But by our claim: $r(C_k) \leq d(s_i, s_j) \leq d(s_i, c^*) + d(s_j, c^*) \leq 2 \cdot r(C^*)$
  - Done!

Hardness of Approximation

- Best polynomial time approximation?
  - Good greedy gives 2-approximation in polynomial time
  - Can we get a better approximation?

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.

- How do we prove this?

Hardness of Approximation

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.
- How do we prove this?
  - Same reduction idea:
    - Show that if there is a polytime algorithm which gives ρ-apx to k-center for some $\rho < 2$, then using this algorithm, we can solve a known NP-complete problem in polytime.
    - Hmm. Which NP-complete problem should we use?
      - How about FriendlyRepresentatives problem from assignment 3?

Hardness of Approximation

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.
- Proof:
  - Consider an instance of FriendlyRepresentatives
    - Given a set of people N, a friendship relation F, and an integer m, we want to check if there exists a subset $S \subseteq N$ of m people such that every person not in S is friends with someone in S.
    - Denote this by (N, F, m)

Hardness of Approximation

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.
- Proof:
  - Consider an instance (N, F, m) of FriendlyRepresentatives
  - Create an instance of k-Center as follows
    - Create a site s_i for each person $i \in N$
    - Define $d(s_i, s_j) = 1$ if $(i, j) \in F$ and 2 if $(i, j) \notin F$
      - Check that this satisfies triangle inequality
    - Set $k = m$
    - Note: There are no other points in this metric space, so you must place centers on sites.

Hardness of Approximation

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.
- Proof:
  - C is a set of friendly representatives if and only if $r(C) = 1$
    - Every center is obviously at distance 0 from itself
    - Every non-center s_j is at distance at most 1 from some $s_i \in C$ if and only if every person not in C is friends with someone in C
  - There are only two possibilities:
    - YES: There exists C with $r(C) = 1$
    - NO: Every C has $r(C) = 2$

Hardness of Approximation

- Theorem: Unless P=NP, there is no polynomial time algorithm which gives ρ-approximation for the k-center problem for $\rho < 2$.
- Proof:
  - YES: There exists C with $r(C) = 1$
    - Since our algorithm gives ρ-approximation with $\rho < 2$, it must return a set C with $r(C) < 2$
    - But $r(C) < 2$ means that $r(C) = 1$
    - So the algorithm returns C with $r(C) = 1$
  - NO: Our algorithm returns a C with $r(C) = 2$
  - So checking $r(C)$ of the C returned by algorithm allows solving FriendlyRepresentatives!

Weighted Set Packing

Weighted Set Packing

- Problem
  - Input: A collection of sets $\mathcal{S} = \{S_1, \dots, S_n\}$ with values $v_1, \dots, v_n \geq 0$. There are *m* set elements.
  - Output: Pick disjoint sets with maximum total value, i.e. pick $W \subseteq \{1, \dots, n\}$ to maximize $\sum_{i \in W} v_i$ subject to the constraint that for all $i, j \in W$, $S_i \cap S_j = \emptyset$.

- This is known to be an NP-hard problem
- *It is also known that for any constant $\epsilon > 0$, you cannot get $O\left(m^{1/2-\epsilon}\right)$ approximation in polynomial time unless NP=ZPP (widely believed to be not true)*

Greedy Template

- *Sort the sets in some order, consider them one-by-one, and take any set that you can along the way.*

- **Greedy Algorithm:**
  - Sort the sets in a specific order.
  - Relabel them as $1,2,\ldots,n$ in this order.
  - $W \leftarrow \emptyset$
  - For $i = 1,\ldots,n$:
    - If $S_i \cap S_j = \emptyset$ for every $j \in W$, then $W \leftarrow W \cup \{i\}$
  - Return W.

Greedy Algorithm

- What order should we sort the sets by?
- We want to take sets with high values.
  - $v_1 \geq v_2 \geq \cdots \geq v_n$? Only m-approximation ☹
- We don't want to exhaust many items too soon.
  - $\frac{v_1}{|S_1|} \geq \frac{v_2}{|S_2|} \geq \cdots \frac{v_n}{|S_n|}$? Also m-approximation ☹
- $\sqrt{m}$-approximation : $\frac{v_1}{\sqrt{|S_1|}} \geq \frac{v_2}{\sqrt{|S_2|}} \geq \cdots \frac{v_n}{\sqrt{|S_n|}}$?

[Lehmann et al. 2011]

Proof of Approximation

- Definitions
  - OPT = Some optimal solution
  - W = Solution returned by our greedy algorithm
  - For $i \in W$,
$$OPT_i = \{j \in OPT, j \geq i : S_i \cap S_j \neq \emptyset\}$$
    OPTi has future j in OPT blocked for inclusion in greedy W because of choosing I (i is also in OPTi).
- Claim 1: $OPT \subseteq \bigcup_{i \in W} OPT_i$

If j from OPT is in W => j in OPTj, else j must be in some OPTi or the greedy algorithm would have chosen it.

- Claim 2: It is enough to show that $\forall i \in W$
$$\sqrt{m} \cdot v_i \geq \sum_{j \in OPT_i} v_j$$

The value of greedy choice i is at least as good 1/ $\sqrt{m}$ * the values from the optimal solution it blocks, and all elements of OPT will be accounted for by the union of OPTi's.

- Observation: For $j \in OPT_i$, $v_j \leq v_i \cdot \frac{\sqrt{|S_j|}}{\sqrt{|S_i|}}$

Greedy ordering.

Proof of Approximation

- Summing over all $j \in OPT_i$:

$$\Sigma_{j \in OPT_i} v_j \le \frac{v_i}{\sqrt{|S_i|}} \cdot \Sigma_{j \in OPT_i} \sqrt{|S_j|}$$

- Using Cauchy-Schwarz ($\Sigma_i x_i y_i \le \sqrt{\Sigma_i x_i^2} \cdot \sqrt{\Sigma_i y_i^2}$)

$$\Sigma_{j \in OPT_i} \sqrt{1. |S_j|} \le \sqrt{|OPT_i|} \cdot \sqrt{\Sigma_{j \in OPT_i} |S_j|}$$

$$\le \boxed{\sqrt{|S_i|} \cdot \sqrt{m}}$$

Every element in Si can block at most one set in OPT => OPTi <= Si.

Also note every Sj in OPTi is disjoint because it belongs to OPT. so the sum of these Sj's is at most m.

Local Search Paradigm

Local Search

- A heuristic paradigm for solving complex problems
  - Sometimes it might provably return an optimal solution
  - But even if not, it might give a good approximation

- Idea:
  - Start with some solution S
  - While there is a "better" solution S' in the **local neighborhood** of S
  - Switch to S'

- Need to define what is "better" and what is a "local neighborhood"

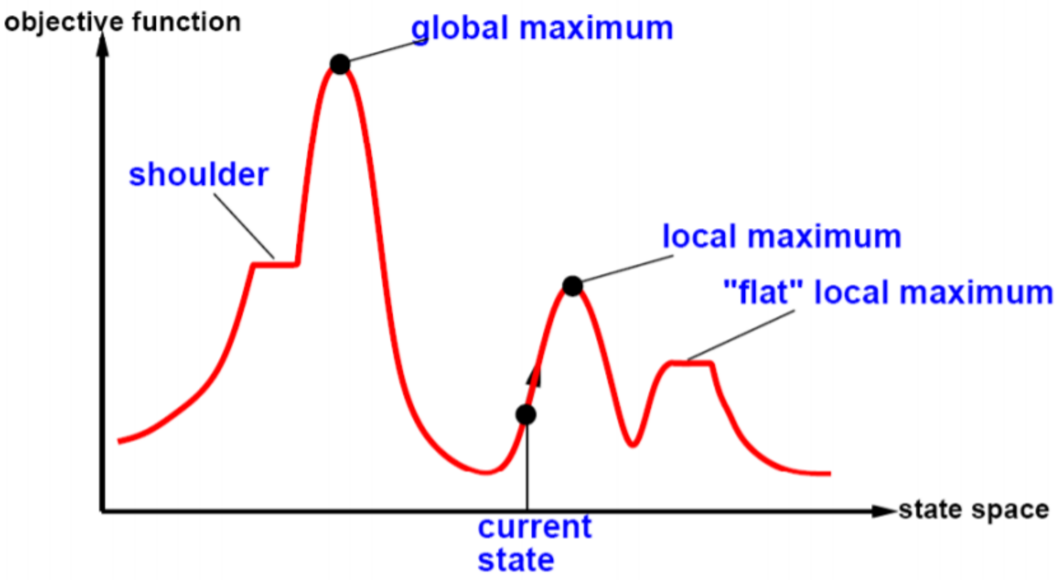
Local Search

- Sometimes local search provably returns an optimal solution

- We already saw such an example: network flow
  - Start with zero flow
  - "Local neighborhood"
    - Set of all flows which can be obtained by augmenting the current flow along a path in the residual graph
  - "Better"
    - Higher flow value

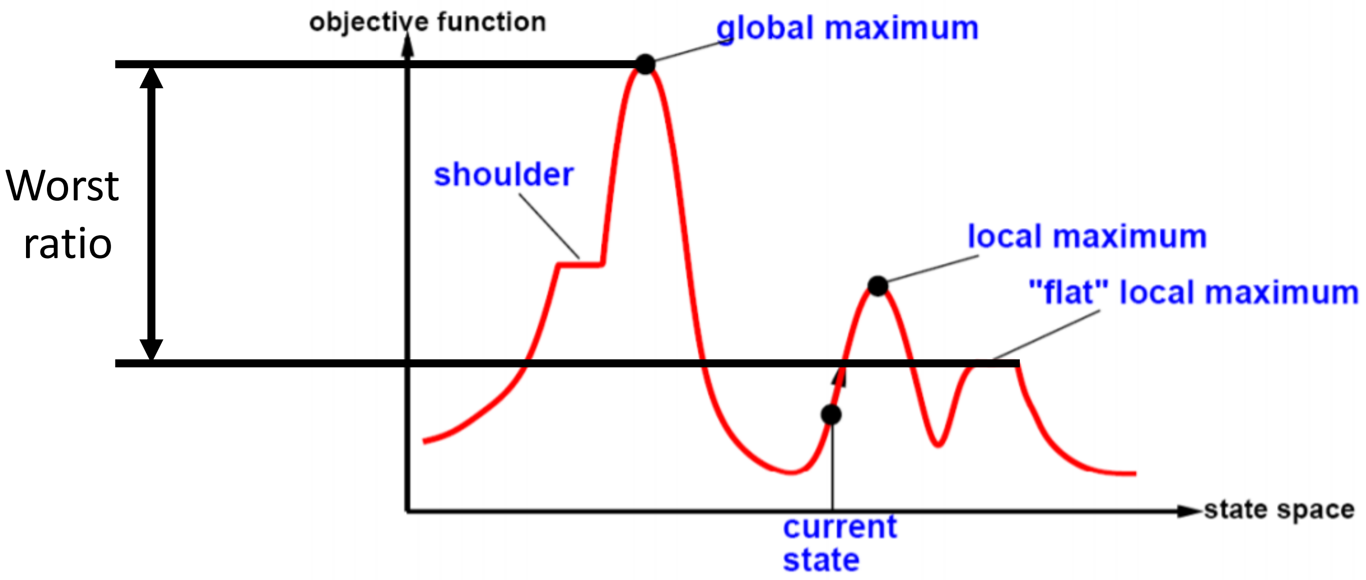
Local Search

- But sometimes it doesn't return an optimal solution, and "gets stuck" in a local maxima



Local Search

- In that case, we want to bound the ratio between the optimal solution and the worst solution local search might return



Max-Cut

Max-Cut

- Problem
  - Input: An undirected graph $G = (V, E)$
  - Output: A partition (A, B) of V that maximizes the number of edges going across the cut, i.e., maximizes $|E'|$ where $E' = \{(u, v) \in E \mid u \in A, v \in B\}$

- This is also known to be an NP-hard problem

- What is a natural local search algorithm for this problem?
  - Given a current partition, what small change can you do to improve the objective value?

Max-Cut

- Local Search
  - Initialize (A, B) arbitrarily.
  - While there is a vertex u such that moving u to the other side improves the objective value:
    - Move u to the other side.

- When does moving u, say from A to B, improve the objective value?
  - When u has more incident edges going within the cut than across the cut, i.e., when $|\{(u,v) \in E \mid v \in A\}| > |\{(u,v) \in E \mid v \in B\}|$

Max-Cut

- **Local Search**
  - Initialize (A, B) arbitrarily.
  - While there is a vertex u such that moving u to the other side improves the objective value:
    - Move u to the other side.

- Why does the algorithm stop?
  - Every iteration increases the number of edges across the cut by at least 1, so the algorithm must stop in at most $|E|$ iterations

Max-Cut

- Local Search
  - Initialize (A, B) arbitrarily.
  - While there is a vertex u such that moving u to the other side improves the objective value:
    - Move u to the other side.

- Approximation ratio?
  - At the end, every vertex has at least as many edges going across the cut as within the cut
  - Hence, at least half of all edges must be going across the cut
    - Exercise: Prove this formally by writing equations.

Weighted Max-Cut

- Variant
  - Now we're given integral edge weights $w: E \to \mathbb{N}$
  - The goal is to maximize the total *weight* of edges going across the cut

- Algorithm
  - The same algorithm works, but now we move u to the other side if the total *weight* of its incident edges going within the cut is greater than the total *weight* of its incident edges going across the cut

Weighted Max-Cut

- Number of iterations?
  - In the unweighted case, we said that the number of edges going across the cut must increase by at least 1, so it takes at most $|E|$ iterations
  - In the weighted case, the total weight of edges going across the cut increases by at least 1, but this could take up to $\sum_{e \in E} w_e$ iterations, which is *exponential* in the input length
    - There are examples where the local search actually takes exponentially many steps

Weighted Max-Cut

- Number of iterations?
  - But we can find a $2 + \epsilon$ approximation in time polynomial in the input length and $\frac{1}{\epsilon}$
  - The idea is to only move vertices when it "sufficiently improves" the objective value

Weighted Max-Cut

- Better approximations?

  - Theorem [Goemans-Williamson]: There exists a polynomial time algorithm for max-cut with approximation ratio $\frac{2}{\pi} \cdot \min_{0 \leq \theta \leq \pi} \frac{\theta}{1-\cos\theta} \approx 0.878$
    - Uses "semidefinite programming" and "randomized rounding"
    - Note: The literature from here on uses approximation ratios ≤ 1, so we will follow that convention in the remaining slides.

  - If the unique games conjecture is true, then this is tight

Exact Max-k-SAT

Exact Max-k-SAT

- Problem
  - Input: An exact k-SAT formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i has exactly k literals, and a weight $w_i \geq 0$ of each clause C_i
  - Output: A truth assignment τ maximizing the number (or total weight) of clauses satisfied under τ

- Let us denote by $W(\tau)$ the total weight of clauses satisfied under τ

- What is a good definition of "local neighborhood"?

Exact Max-k-SAT

- Local neighborhood:
  - $N_d(\tau)$ = set of all truth assignments which can be obtained by changing the value of at most d variables in τ

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.

Exact Max-k-SAT

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - Let τ be a local optimum
    - S_0 = set of clauses not satisfied under τ
    - S_1 = set of clauses from which exactly one literal is true under τ
    - S_2 = set of clauses from which both literals are true under τ
    - $W(S_0), W(S_1), W(S_2)$ be the corresponding total weights
    - Goal: $W(S_1) + W(S_2) \geq {}^{2}/_{3} \cdot \left(W(S_0) + W(S_1) + W(S_2)\right)$
      - Equivalently, $W(S_0) \leq {}^{1}/_{3} \cdot \left(W(S_0) + W(S_1) + W(S_2)\right)$

Exact Max-k-SAT

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - Let τ be a local optimum
    - S_0 = set of clauses not satisfied under τ
    - S_1 = set of clauses from which exactly one literal is true under τ
    - S_2 = set of clauses from which both literals are true under τ
    - $W(S_0), W(S_1), W(S_2)$ be the corresponding total weights
    - Goal: $W(S_1) + W(S_2) \geq {}^{2}/_{3} \cdot \big(W(S_0) + W(S_1) + W(S_2)\big)$
      - Equivalently, $W(S_0) \leq {}^{1}/_{3} \cdot \big(W(S_0) + W(S_1) + W(S_2)\big)$

Exact Max-k-SAT

- **Theorem:** The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- **Proof:**
  - Clause C involves variable j if it contains x_j or $\overline{x_j}$
    - A_j = set of clauses in S_0 involving variable j
    - B_j = set of clauses in S_1 involving variable j such that it is the literal of variable j that is true under τ
    - C_j = set of clauses in S_2 involving variable j
    - $W(A_j), W(B_j), W(C_j)$ be the corresponding total weights

Exact Max-k-SAT

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - $2\, W(S_0) = \sum_j W(A_j)$
    - Every clause in S_0 is counted twice on the RHS
  - $W(S_1) = \sum_j W(B_j)$
    - Every clause in S_1 is only counted once on the RHS for the variable whose literal was true under τ
  - For each j : $W(A_j) \leq W(B_j)$
    - From local optimality of τ, since otherwise flipping the truth value of variable j would have increased the total weight

Exact Max-k-SAT

- Theorem: The local search with $d = 1$ gives a ${}^{2}/_{3}$ approximation to Exact Max-2-SAT.
- Proof:
  - $2\,W(S_0) \leq W(S_1)$
    - Summing the third equation on the last slide over all j, and then using the first two equations on the last slide
  - Hence:
    - $3\,W(S_0) \leq W(S_0) + W(S_1) \leq W(S_0) + W(S_1) + W(S_2)$
    - Precisely the condition we wanted to prove…

Exact Max-k-SAT

- Higher d?
  - Searches over a larger neighborhood
  - May get a better approximation ratio, but increases the running time as we now need to check if any neighbor in a large neighborhood provides a better objective
  - The bound is still $2/3$ for $d = o(n)$
  - It is no better than $4/5$ for $d < {}^{n}/_{2}$
  - It can be shown that with $d = {}^{n}/_{2}$, the algorithm always terminates at an optimal solution

Exact Max-k-SAT

- Better approximation?
  - We can learn something from our proof
  - Note that we did not use anything about $W(S_2)$, and simply added it at the end
  - If we could also guarantee that $W(S_0) \leq W(S_2)$…
    - Then we would get $4\, W(S_0) \leq W(S_0) + W(S_1) + W(S_2)$, which would give a $^3/_4$ approximation
  - Result (without proof): This can be done by including just one more assignment in the neighborhood: $N(\tau) = N_1(\tau) \cup \{\tau^c\}$, where τ^c = complement of τ

Exact Max-k-SAT

- What if we do not want to modify the neighborhood?
  - A slightly different tweak also works
  - We want to weigh clauses in $W(S_2)$ more because when we get a clause through S_2, we get more robustness (it can withstand changes in single variables)

- Modified local search:
  - Start at arbitrary τ
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

Exact Max-k-SAT

- Modified local search:
  - Start at arbitrary τ
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

- Note:
  - This is the first time that we're using a definition of "better" in local search paradigm that does not quite align with the ultimate objective we want to maximize
  - This is called "non-oblivious local search"

Exact Max-k-SAT

- Modified local search:
  - Start at arbitrary τ
  - While there is an assignment in $N_1(\tau)$ that improves the potential $1.5\, W(S_1) + 2\, W(S_2)$
    - Switch to that assignment

- Result (without proof):
  - Modified local search gives $^3/_4$-approximation to Exact Max-2-SAT

Exact Max-k-SAT

- More generally:
  - The same technique works for higher values of k
  - Gives $\frac{2^k-1}{2^k}$ approximation for Exact Max-k-SAT
    - We'll see how to achieve the same approximation using a much simpler technique

- Note: This is ${}^7/_8$ for Exact Max-3-SAT
  - Theorem [Håstad]: Achieving ${}^7/_8 + \epsilon$ approximation where $\epsilon > 0$ is NP-hard.
    - Uses PCP (probabilistically checkable proofs) technique