More Randomized Algorithms

Randomization So Far

- Analyzed simple probabilistic processes
  - Birthday paradox
  - Pokemon collector problem
  - Random walks
- Designing and analyzed simple randomized algorithms
  - Karger's min cut
  - Randomized selection
  - Randomized quicksort
- Today: Use randomization to design approximation algorithms for NP complete problems:
  - Max-3-SAT and Max cut

Admin

- Next class in a week!
- Assignment 9 out Friday
- Any questions?

Randomized Approximation Algorithms

- Sometimes it's hard to get the correct answer to a problem using an efficient algorithm
- Can we give guarantees on the algorithm's performance, even if they fall short of giving the correct answer?
- **Approximation Algorithm**: gives an answer with a guarantee of the quality of that answer compared to the optimal answer

- First example:
- We can't satisfy all clauses of a 3-SAT instance in polynomial time. If the optimal algorithm satisfies k clauses in the 3SAT instance, how many can we satisfy in polynomial time?
    - $7k/8$
- We'll use a randomized algorithm to get this bound

Randomized Approximation:
Max 3-SAT

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \dots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \lor x_5 \lor x_6) \land \dots$$

A “clause” in 3-SAT consists of 3 variables, at least one of which must be true

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \ldots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

- What if we:

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \ldots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

- What if we:
    - Flip a fair coin for each variable

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \ldots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

- What if we:
  - Flip a fair coin for each variable
  - If heads, set variable to true

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \ldots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

- What if we:
  - Flip a fair coin for each variable
  - If heads, set variable to true
  - If tails, set variable to false

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.
- What if we:
    - Flip a fair coin for each variable
    - If heads, set variable to true
    - If tails, set variable to false

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \dots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

Max 3-SAT

- **Maximum 3-satisfiability.** Given a 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.
- **Remark.** NP-hard problem.

$$(\neg x_3 \vee x_5 \vee x_6) \wedge \ldots$$

A "clause" in 3-SAT consists of 3 variables, at least one of which must be true

- What if we:
  - Flip a fair coin for each variable
  - If heads, set variable to true
  - If tails, set variable to false
- What is the expected number of clauses satisfied by a random assignment?

Max 3-SAT

- **Claim**. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$
- **Proof**.

Max 3-SAT

- **Claim**. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$
- **Proof**.
  - Define indicate random variables $Z_i = 1$ if clause C_i is satisfied, and zero otherwise

Max 3-SAT

- **Claim**. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$
- **Proof**.
  - Define indicate random variables $Z_i = 1$ if clause C_i is satisfied, and zero otherwise
  - Let Z be random variable equal to the # of clauses satisfied

Max 3-SAT

- **Claim**. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$
- **Proof**.
  - Define indicate random variables $Z_i = 1$ if clause C_i is satisfied, and zero otherwise
  - Let Z be random variable equal to the # of clauses satisfied
  - $E[Z] = E[\sum_{i=1}^{k} Z_i] = \sum_{i=1}^{n} E[Z_i]$

Max 3-SAT

- **Claim**. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is $7k/8$
- **Proof**.
  - Define indicate random variables $Z_i = 1$ if clause C_i is satisfied, and zero otherwise
  - Let Z be random variable equal to the # of clauses satisfied
  - $E[Z] = E[\sum_{i=1}^{k} Z_i] = \sum_{i=1}^{n} E[Z_i]$
  - $E[Z_i] =$ Pr[clause C_i is satisfied] =

    $1-$Pr[clause C_i is not satisfied]

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
    - If each variable in C_i is set so that its literal evaluates to false

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
  - If each variable in C_i is set so that its literal evaluates to false
  - Each variable's truth assignment is set independently

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
    - If each variable in C_i is set so that its literal evaluates to false
    - Each variable's truth assignment is set independently
    - Probability of this happening is $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
    - If each variable in C_i is set so that its literal evaluates to false
    - Each variable's truth assignment is set independently
    - Probability of this happening is $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$
- Probability that clause C_i is not satisfied is thus $\frac{1}{8}$

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
    - If each variable in C_i is set so that its literal evaluates to false
    - Each variable's truth assignment is set independently
    - Probability of this happening is $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$
- Probability that clause C_i is not satisfied is thus $\frac{1}{8}$
    - $E[Z_i] =$ Pr[clause C_i is satisfied] $= \frac{7}{8}$

Probability Clause is Not Satisfied

- Let C_i be an arbitrary clause
- When is C_i not satisfied?
  - If each variable in C_i is set so that its literal evaluates to false
  - Each variable's truth assignment is set independently
  - Probability of this happening is $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$
- Probability that clause C_i is not satisfied is thus $\frac{1}{8}$
- $E[Z_i] = \Pr[\text{clause } C_i \text{ is satisfied}] = \frac{7}{8}$
- $E[Z] = \sum_{i=1}^{k} \frac{7}{8} = \frac{7k}{8}$

Surprising Conclusion

- Expected number of clauses satisfied is thus $E[Z] = \frac{7k}{8}$
- A random variable is at least its expectation some of the time

Surprising Conclusion

- Expected number of clauses satisfied is thus $E[Z] = \frac{7k}{8}$
- A random variable is at least its expectation some of the time
- **Conclusion**. For every instance of 3-SAT, there is a truth assignment that satisfies at least a $7/8$th fraction of clauses.

Surprising Conclusion

- Expected number of clauses satisfied is thus $E[Z] = \frac{7k}{8}$
- A random variable is at least its expectation some of the time
- **Conclusion**. For every instance of 3-SAT, there is a truth assignment that satisfies at least a $7/8$th fraction of clauses.
- This is a non-obvious fact about 3-SAT—the existence of an assignment satisfying that many clauses—whose statement has nothing to do with the randomization that led us to it

Surprising Conclusion

- Expected number of clauses satisfied is thus $E[Z] = \dfrac{7k}{8}$
- A random variable is at least its expectation some of the time
- **Conclusion**. For every instance of 3-SAT, there is a truth assignment that satisfies at least a $7/8$th fraction of clauses.
- This is a non-obvious fact about 3-SAT—the existence of an assignment satisfying that many clauses—whose statement has nothing to do with the randomization that led us to it
- Widespread principle in combinatorics:

Surprising Conclusion

- Expected number of clauses satisfied is thus $E[Z] = \dfrac{7k}{8}$
- A random variable is at least its expectation some of the time
- **Conclusion**. For every instance of 3-SAT, there is a truth assignment that satisfies at least a $7/8$th fraction of clauses.
- This is a non-obvious fact about 3-SAT—the existence of an assignment satisfying that many clauses—whose statement has nothing to do with the randomization that led us to it
- Widespread principle in combinatorics:
    - **Probabilistic method**. [Paul Erdös] Prove the existence of a non-obvious property by showing that a random construction produces it with positive probability!

(7/8)-Approximation: Las Vegas

- Can we turn this into an approximation algorithm that is guaranteed to return a truth assignment that satisfies at least $7/8$th clauses
  - But has expected running time that is polynomial
  - That is, a Las Vegas style approximation algorithm

(7/8)-Approximation: Las Vegas

- Can we turn this into an approximation algorithm that is guaranteed to return a truth assignment that satisfies at least $7/8$th clauses
    - But has expected running time that is polynomial
    - That is, a Las Vegas style approximation algorithm
- **Simple and standard trick.** Repeat until you get what you are looking for: that is, randomly generate truth assignments until one of them satisfies at least $7/8$th of the clauses.

(7/8)-Approximation: Las Vegas

- Can we turn this into an approximation algorithm that is guaranteed to return a truth assignment that satisfies at least $7/8$th clauses
  - But has expected running time that is polynomial
  - That is, a Las Vegas style approximation algorithm
- **Simple and standard trick.** Repeat until you get what you are looking for: that is, randomly generate truth assignments until one of them satisfies at least $7/8$th of the clauses.
- Suppose we can show that the probability that a random assignment satisfies at least $7/8$th of the clauses is at least p

(7/8)-Approximation: Las Vegas

- Can we turn this into an approximation algorithm that is guaranteed to return a truth assignment that satisfies at least $7/8$th clauses
  - But has expected running time that is polynomial
  - That is, a Las Vegas style approximation algorithm
- **Simple and standard trick.** Repeat until you get what you are looking for: that is, randomly generate truth assignments until one of them satisfies at least $7/8$th of the clauses.
- Suppose we can show that the probability that a random assignment satisfies at least $7/8$th of the clauses is at least p
- Then the expected number of tries we need until success is $1/p$

(7/8)-Approximation: Las Vegas

- Can we turn this into an approximation algorithm that is guaranteed to return a truth assignment that satisfies at least $7/8$th clauses
    - But has expected running time that is polynomial
    - That is, a Las Vegas style approximation algorithm
- **Simple and standard trick.** Repeat until you get what you are looking for: that is, randomly generate truth assignments until one of them satisfies at least $7/8$th of the clauses.
- Suppose we can show that the probability that a random assignment satisfies at least $7/8$th of the clauses is at least p
- Then the expected number of tries we need until success is $1/p$
- As long as p is polynomial, expected running time is polynomial

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$th of the clauses is at least $1/(8k)$.
- For $j = 1,2,\ldots,k$, let p_j denote the probability that a random assignment satisfies exactly j clauses

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$th of the clauses is at least $1/(8k)$.
- For $j = 1,2,\ldots,k$, let p_j denote the probability that a random assignment satisfies exactly j clauses
- Expected number of satisfies clauses $E[Z] = \sum_{j=0}^{k} j \cdot p_j = \frac{7}{8}k$

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$th of the clauses is at least $1/(8k)$.
- For $j = 1,2,\ldots,k$, let p_j denote the probability that a random assignment satisfies exactly j clauses
- Expected number of satisfies clauses $E[Z] = \sum_{j=0}^{k} j \cdot p_j = \frac{7}{8}k$
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$th of the clauses is at least $1/(8k)$.
- For $j = 1,2,\ldots,k$, let p_j denote the probability that a random assignment satisfies exactly j clauses
- Expected number of satisfies clauses $E[Z] = \sum_{j=0}^{k} j \cdot p_j = \frac{7}{8}k$
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$th of the clauses is at least $1/(8k)$.
- For $j = 1,2,\ldots,k$, let p_j denote the probability that a random assignment satisfies exactly j clauses
- Expected number of satisfies clauses $E[Z] = \sum_{j=0}^{k} j \cdot p_j = \frac{7}{8}k$
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?
- Rewrite the expectation as:
  $$E[Z] = \frac{7}{8}k = \sum_{j<7k/8} j \cdot p_j + \sum_{j \geq 7k/8} j \cdot p_j$$

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$ ths of the clauses is at least $1/(8k)$.
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$ ths of the clauses is at least $1/(8k)$.
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?
- $E[Z] = \frac{7}{8}k = \sum_{j < 7k/8} j \cdot p_j + \sum_{j \geq 7k/8} j \cdot p_j$

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$ ths of the clauses is at least $1/(8k)$.
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?
- $E[Z] = \frac{7}{8}k = \sum_{j < 7k/8} j \cdot p_j + \sum_{j \geq 7k/8} j \cdot p_j$
- $\leq \sum_{j < 7k/8} j \cdot 1 + \sum_{j \geq 7k/8} k \cdot p_j$

$$\leq \left(\frac{7k}{8} - \frac{1}{8} \right) \cdot 1 + kp$$

Analyzing the Approximation

- **Claim**. Probability that a random assignment satisfies at least $7/8$ ths of the clauses is at least $1/(8k)$.
- Define $p = \sum_{j \geq 7k/8} p_j$. Then $1 - p = \sum_{j < 7k/8} p_j$
- How do we use the expectation to get a lower bound on p?
- $E[Z] = \frac{7}{8}k = \sum_{j < 7k/8} j \cdot p_j + \sum_{j \geq 7k/8} j \cdot p_j$
- $\leq \sum_{j < 7k/8} j \cdot 1 + \sum_{j \geq 7k/8} k \cdot p_j$

$$\leq \left(\frac{7k}{8} - \frac{1}{8}\right) \cdot 1 + kp$$

- Which gives us $p \geq \frac{1}{8k}$

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$
- **Conclusion.**

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$
- **Conclusion.**
- Max-number of clauses that can be satisfied?

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$
- **Conclusion.**
- Max-number of clauses that can be satisfied?
  - k

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$
- **Conclusion.**
- Max-number of clauses that can be satisfied?
  - k
- There is a randomized algorithm with polynomial running time that is $7/8$th approximation algorithm to MAX 3-SAT

(7/8)-Approximation

- Thus with probability at least $1/(8k)$ we succeed in finding an assignment that satisfies at least $7/8$th fraction of the clauses
- How many tries before we are succeed in expectation?
  - $8k$
- **Conclusion.**
- Max-number of clauses that can be satisfied?
  - k
- There is a randomized algorithm with polynomial running time that is $7/8$th approximation algorithm to MAX 3-SAT
- Fun fact: It is **NP hard** to approximation MAX 3-SAT with an approximation factor $7/8 + \varepsilon$, for any $\epsilon > 0$ [Håstad 97]

Randomized Approximation: Max Cut

Max-Cut

- **Global max-cut problem.** Given an undirected graph $G = (V, E)$, find a cut (A, B) of maximum cardinality (that is, max # of edges crossing it).

Max-Cut

- **Global max-cut problem.** Given an undirected graph $G = (V, E)$, find a cut (A, B) of maximum cardinality (that is, max # of edges crossing it).
- Interestingly: many polynomial-time (some randomized) algorithms for the min-cut variant, as we discussed a couple classes ago

Max-Cut

- **Global max-cut problem.** Given an undirected graph $G = (V, E)$, find a cut (A, B) of maximum cardinality (that is, max # of edges crossing it).
- Interestingly: many polynomial-time (some randomized) algorithms for the min-cut variant, as we discussed a couple classes ago
    - But global max-cut is NP-hard

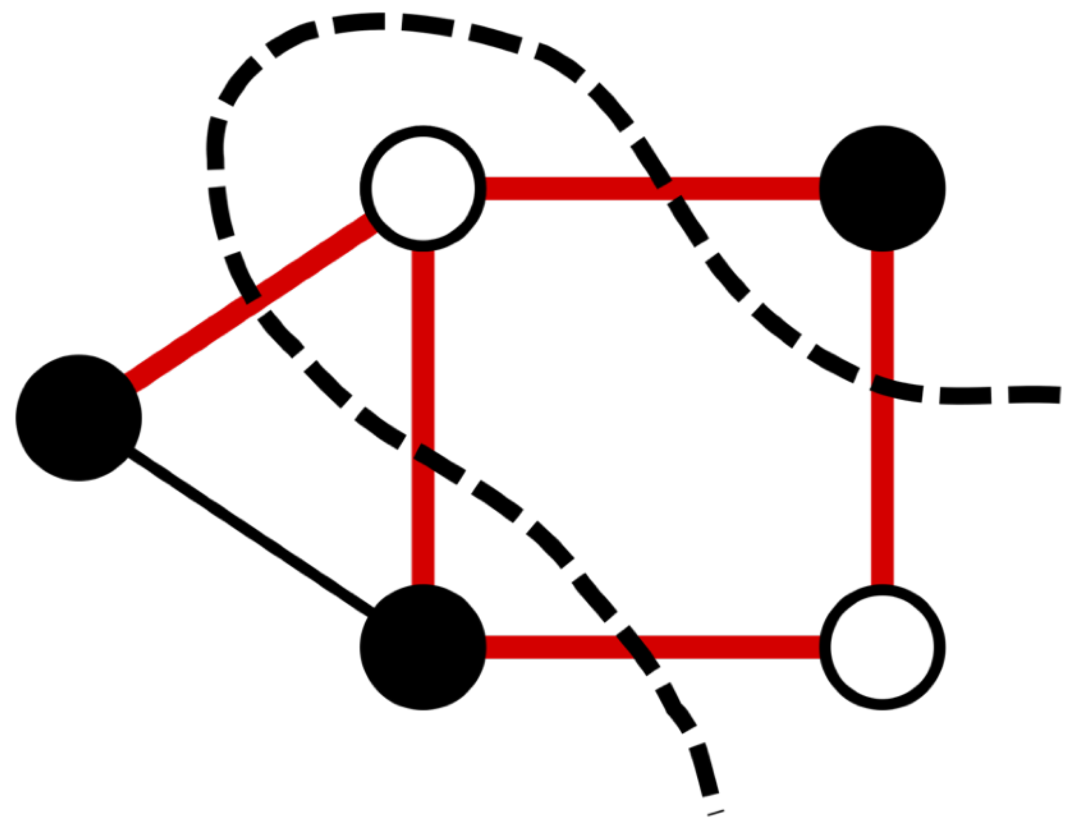
Max-Cut

- **Global max-cut problem.** Given an undirected graph $G = (V, E)$, find a cut (A, B) of maximum cardinality (that is, max # of edges crossing it).
- Interestingly: many polynomial-time (some randomized) algorithms for the min-cut variant, as we discussed a couple classes ago
    - But global max-cut is NP-hard
- We will design an approximation algorithm for this problem using randomization

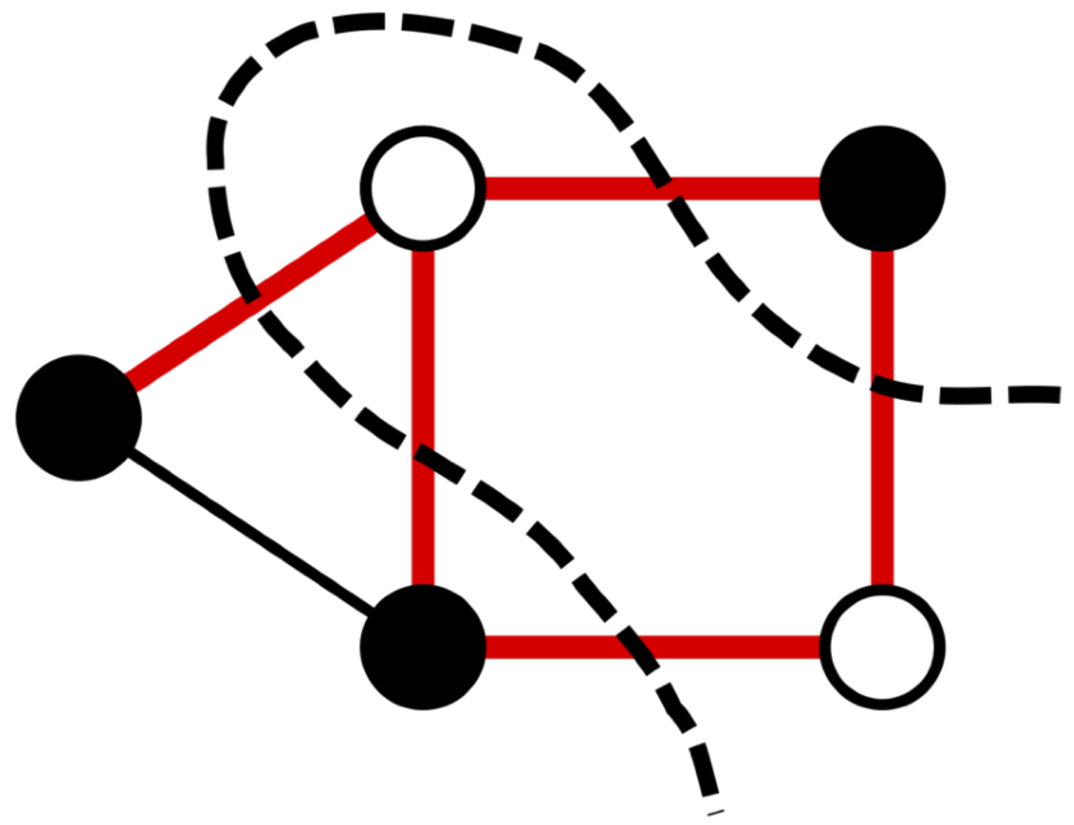
Max-Cut

- **Global max-cut problem.** Given an undirected graph $G = (V, E)$, find a cut (A, B) of maximum cardinality (that is, max # of edges crossing it).
- Interestingly: many polynomial-time (some randomized) algorithms for the min-cut variant, as we discussed a couple classes ago
    - But global max-cut is NP-hard
- We will design an approximation algorithm for this problem using randomization
- A 1/2-approximation to max-cut will produce a cut whose size is at least $1/2$ of the optimal (largest cut in the graph)

Motivation

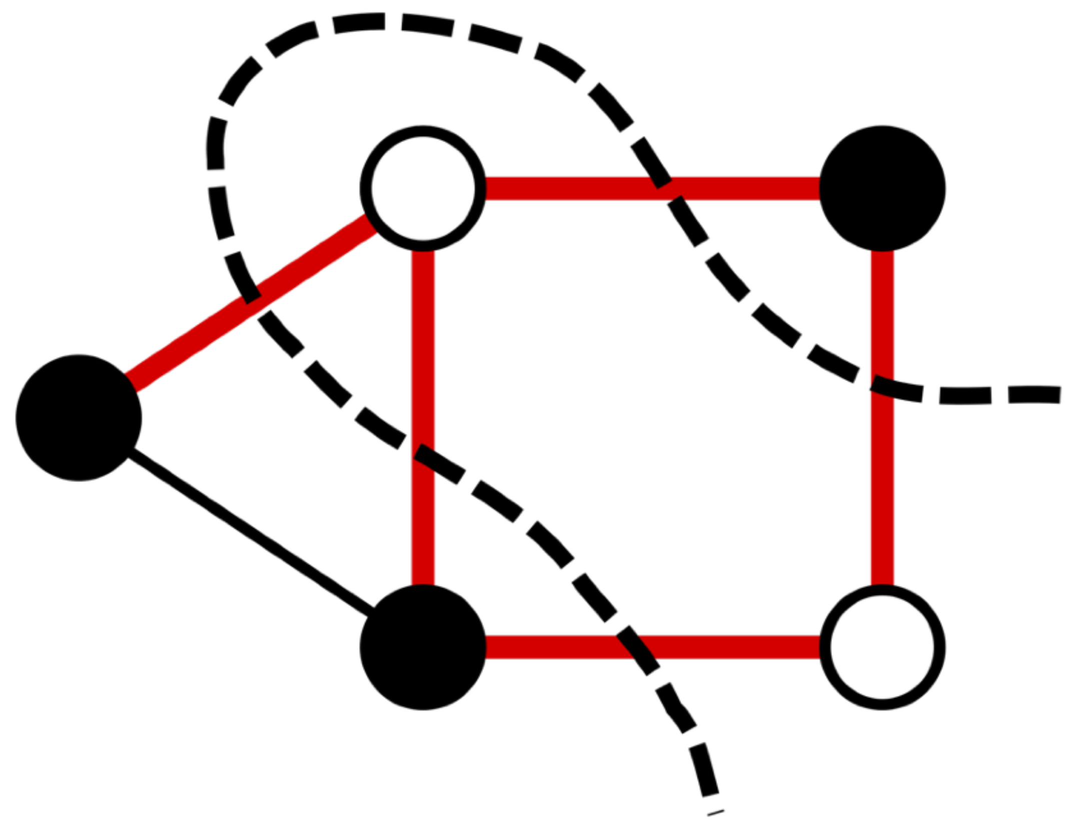


Motivation



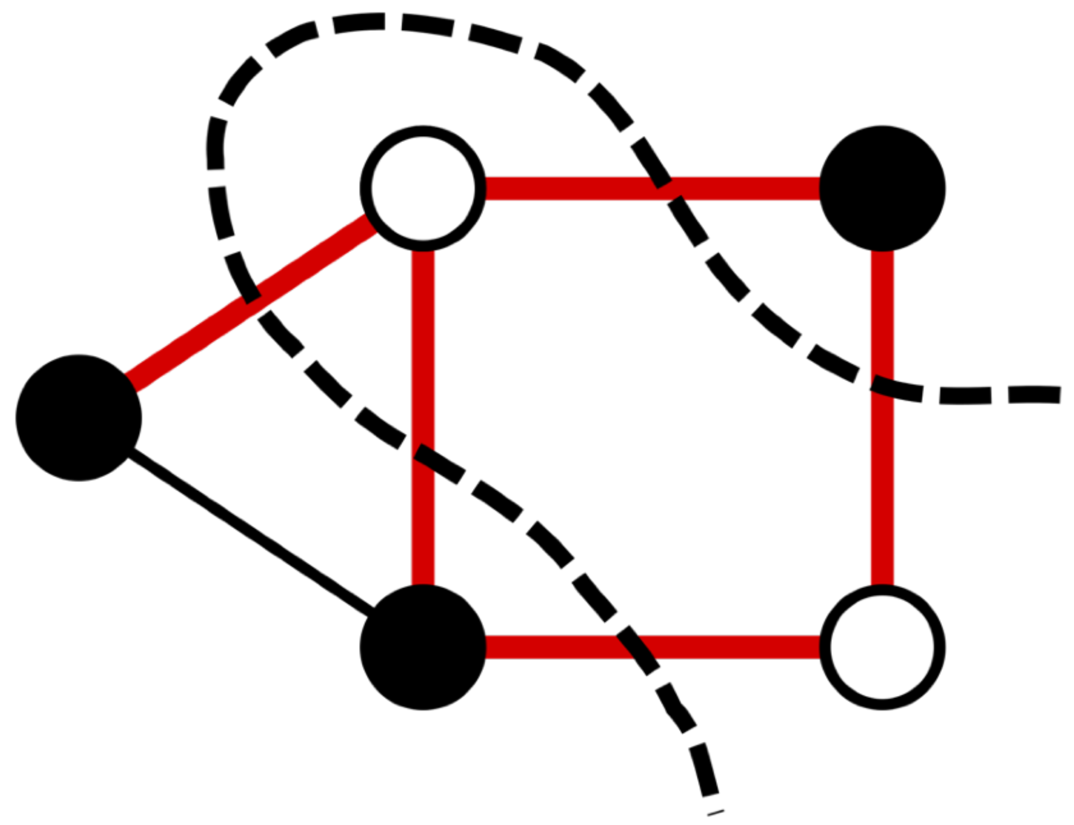
- We're asking to partition the graph into two pieces, minimizing edges within each piece

Motivation



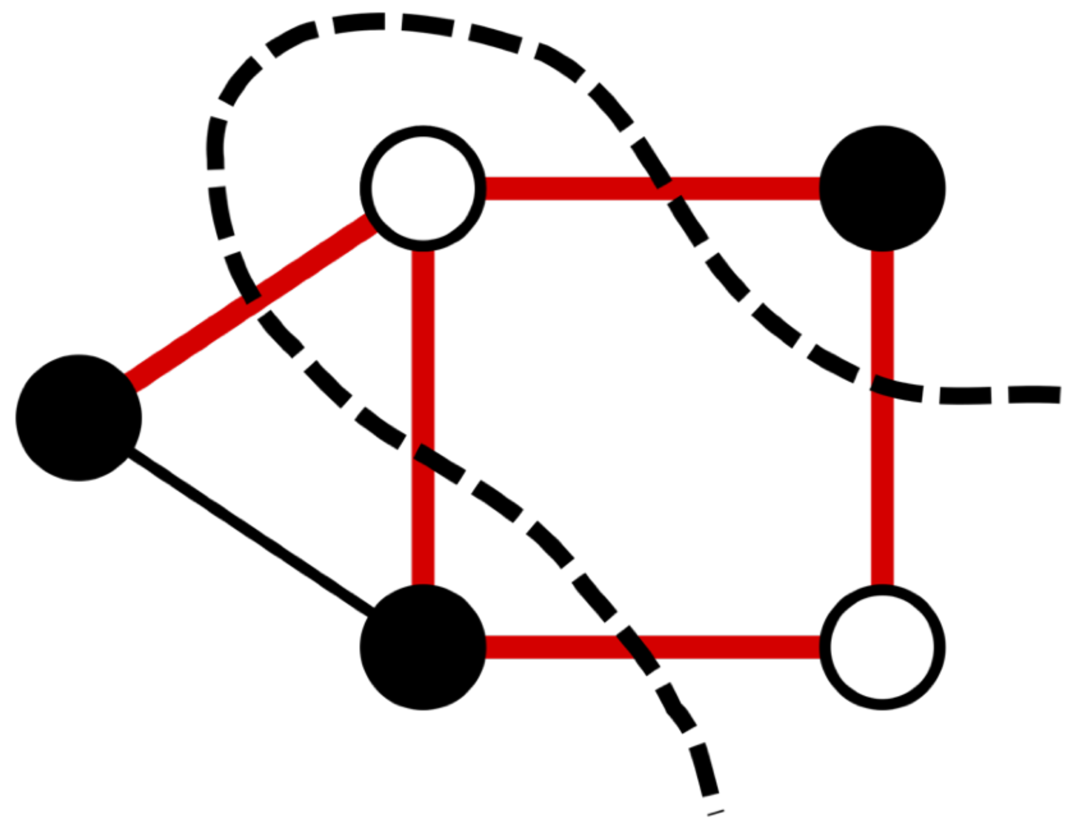
- We're asking to partition the graph into two pieces, minimizing edges within each piece
- If we put an edge between incompatible/different/etc. items, this is like asking us to partition the vertices into two similar/compatible groups

Motivation



- We’re asking to partition the graph into two pieces, minimizing edges within each piece
- If we put an edge between incompatible/different/etc. items, this is like asking us to partition the vertices into two similar/compatible groups
- This (and similar) problems are really important in data science and machine learning

Motivation



- We're asking to partition the graph into two pieces, minimizing edges within each piece
- If we put an edge between incompatible/different/etc. items, this is like asking us to partition the vertices into two similar/compatible groups
- This (and similar) problems are really important in data science and machine learning
- It's a huge pain that this is NP-hard. But, at least we can approximate it!

A Really Simple Algorithm

- For each vertex, toss a fair coin
    - If it lands heads, place the node into one part of the cut
    - If it lands tails, place the node into the other part of the cut

A Really Simple Algorithm

- For each vertex, toss a fair coin
    - If it lands heads, place the node into one part of the cut
    - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?

A Really Simple Algorithm

- For each vertex, toss a fair coin
    - If it lands heads, place the node into one part of the cut
    - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?
- For each edge e let C_e be an indicator random variable where

A Really Simple Algorithm

- For each vertex, toss a fair coin
  - If it lands heads, place the node into one part of the cut
  - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?
- For each edge e let C_e be an indicator random variable where
  - $C_e = 1$ if edge e crosses the cut

A Really Simple Algorithm

- For each vertex, toss a fair coin
  - If it lands heads, place the node into one part of the cut
  - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?
- For each edge e let C_e be an indicator random variable where
  - $C_e = 1$ if edge e crosses the cut
  - $C_e = 0$ otherwise

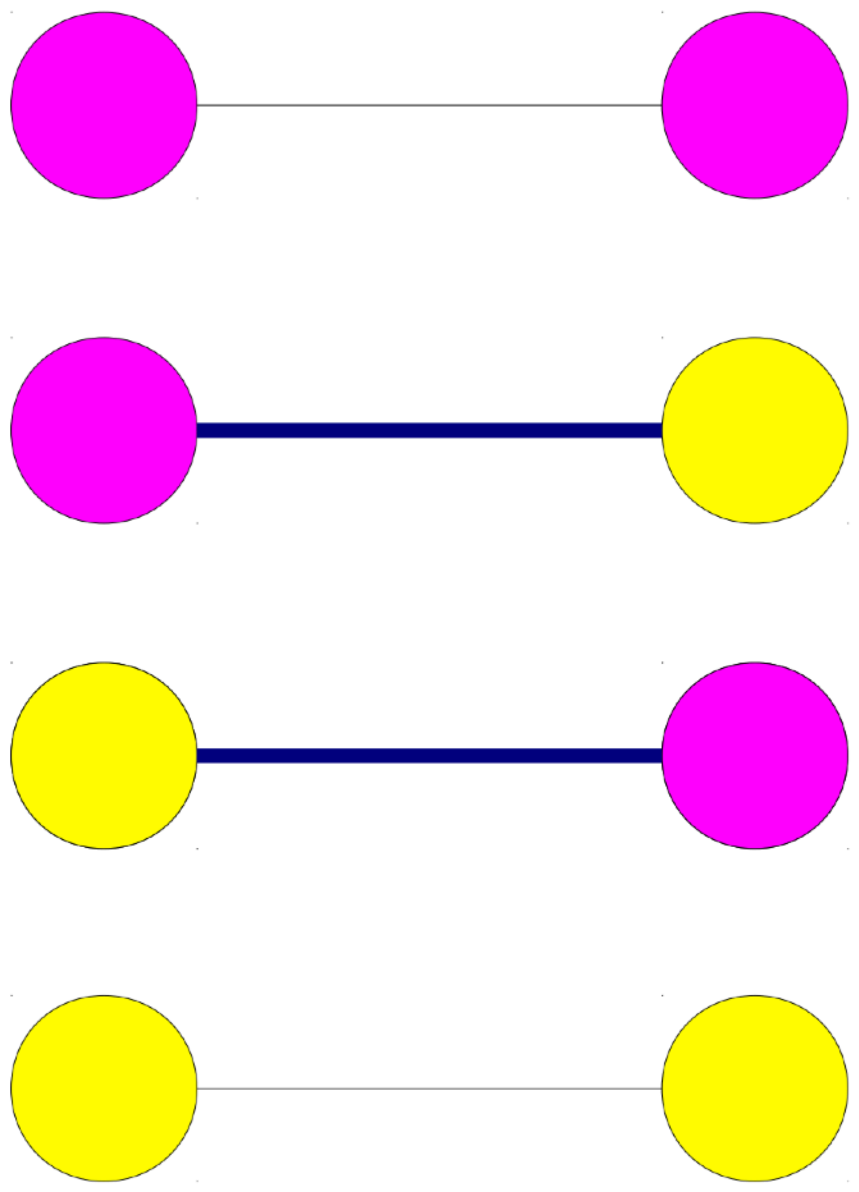
A Really Simple Algorithm

- For each vertex, toss a fair coin
    - If it lands heads, place the node into one part of the cut
    - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?
- For each edge e let C_e be an indicator random variable where
    - $C_e = 1$ if edge e crosses the cut
    - $C_e = 0$ otherwise
- Then the total number of cross edges X of the cut produces is given by the sum of the indicator random variables, we want $E[X]$

A Really Simple Algorithm

- For each vertex, toss a fair coin
  - If it lands heads, place the node into one part of the cut
  - If it lands tails, place the node into the other part of the cut
- **Question.** In expectation, how large of a cut will this algorithm produce?
- For each edge e let C_e be an indicator random variable where
  - $C_e = 1$ if edge e crosses the cut
  - $C_e = 0$ otherwise
- Then the total number of cross edges X of the cut produces is given by the sum of the indicator random variables, we want $E[X]$
  - $X = \sum_{e \in E} C_e$

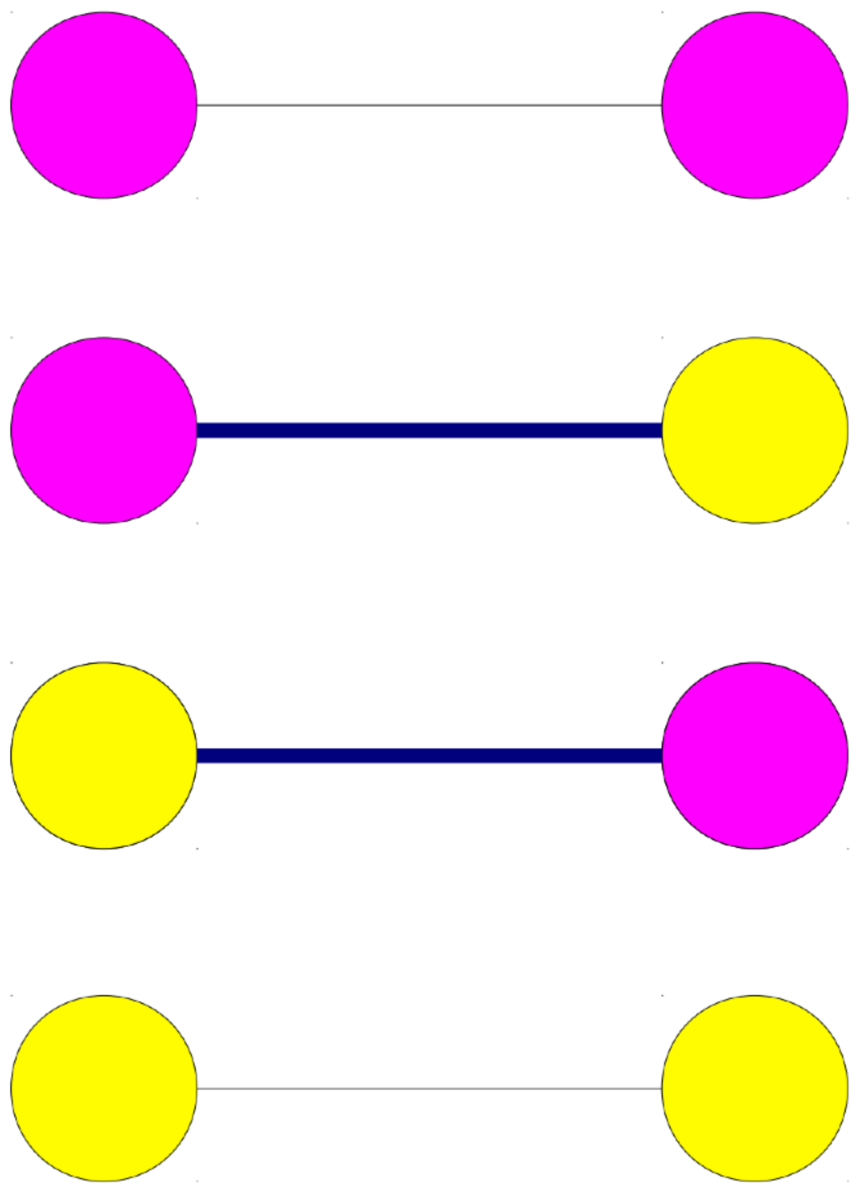
Analyzing the Max-Cut Algorithm



Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$

Analyzing the Max-Cut Algorithm



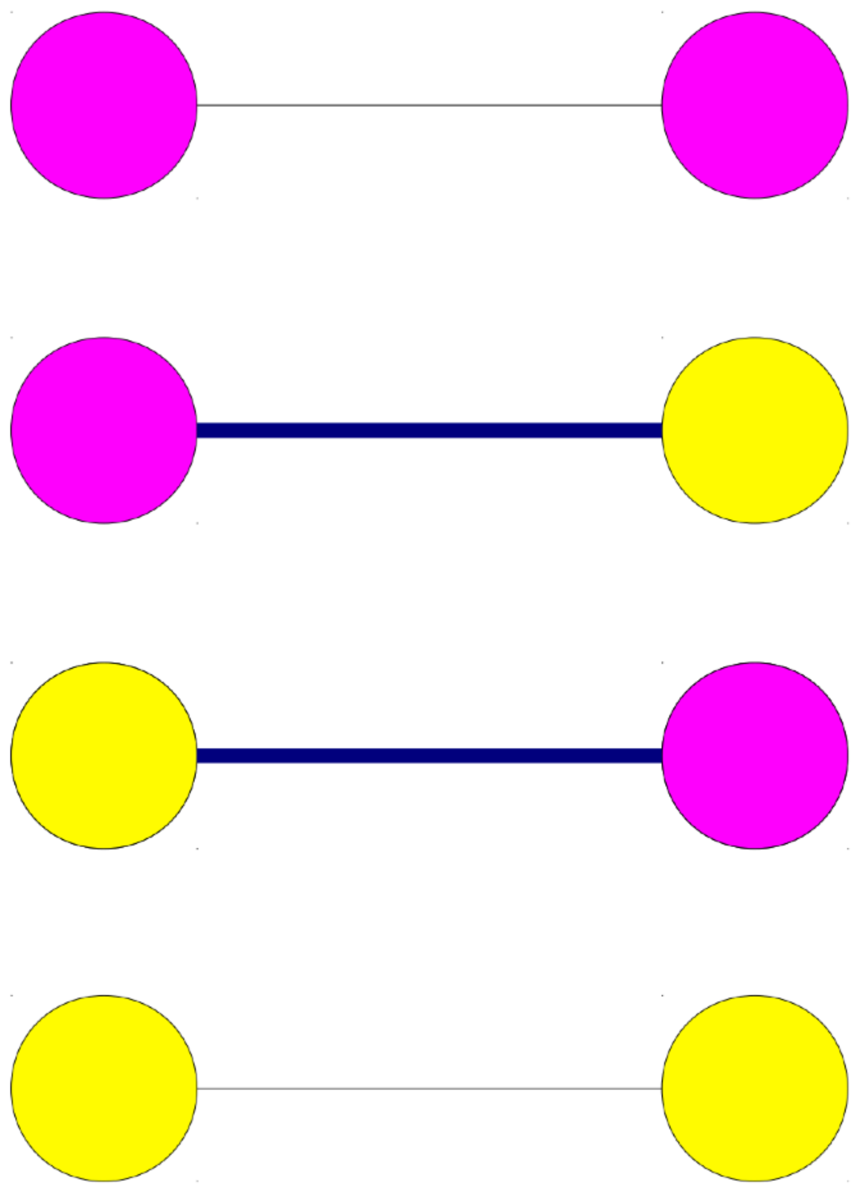
Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$
- $\text{Pr}[e \text{ crosses the cut}] = 1/2$

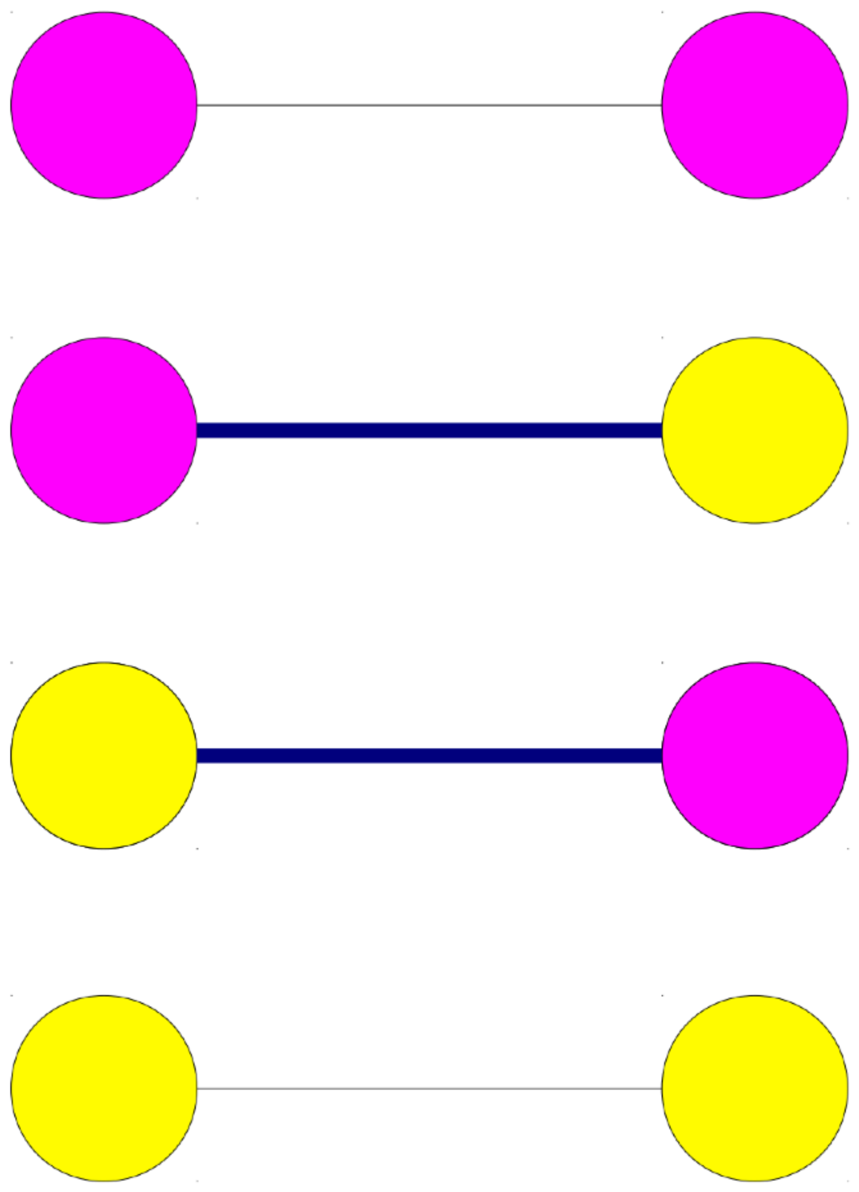
Analyzing the Max-Cut Algorithm

Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$
- $\text{Pr}[e \text{ crosses the cut}] = 1/2$
- $E[X] = \sum_e \frac{1}{2} = \frac{m}{2}$



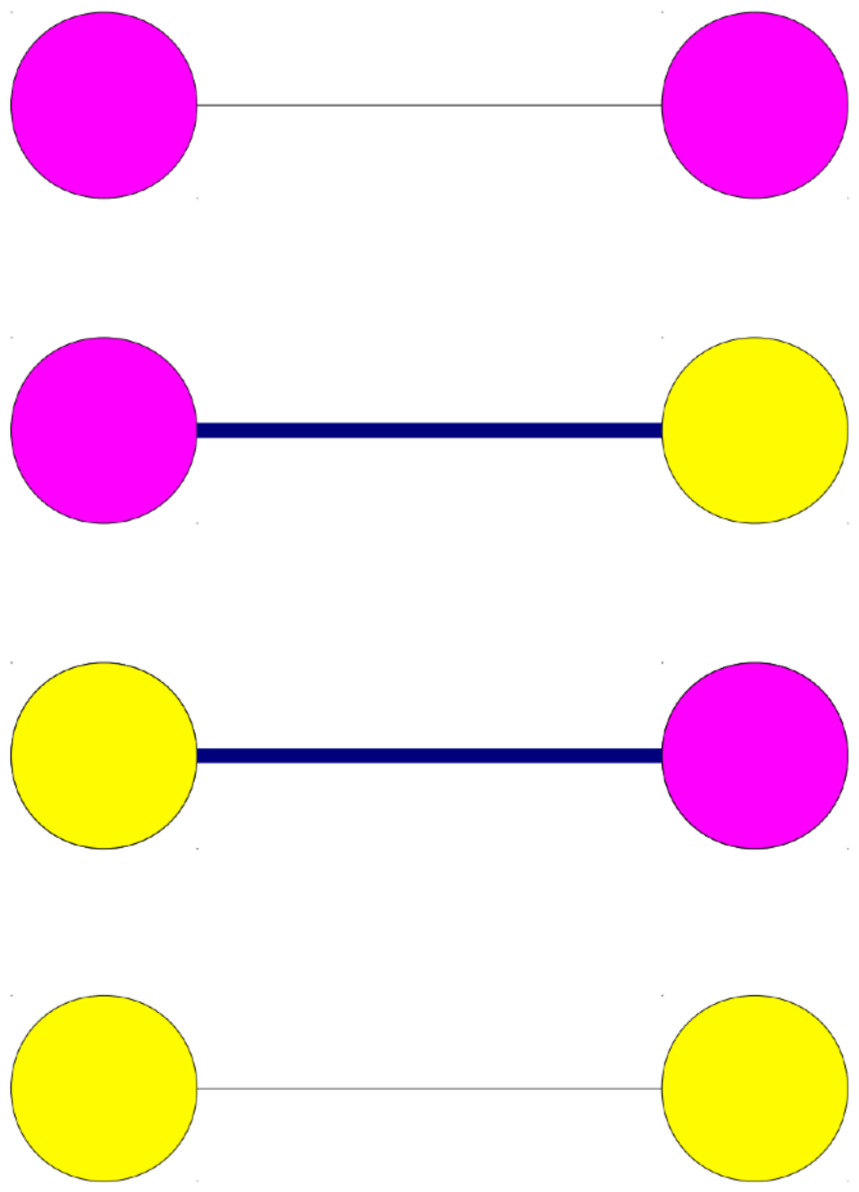
Analyzing the Max-Cut Algorithm



Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$
- $\text{Pr}[e \text{ crosses the cut}] = 1/2$
- $E[X] = \sum_e \frac{1}{2} = \frac{m}{2}$
- What is the maximum number of edges that can cross *any* cut?

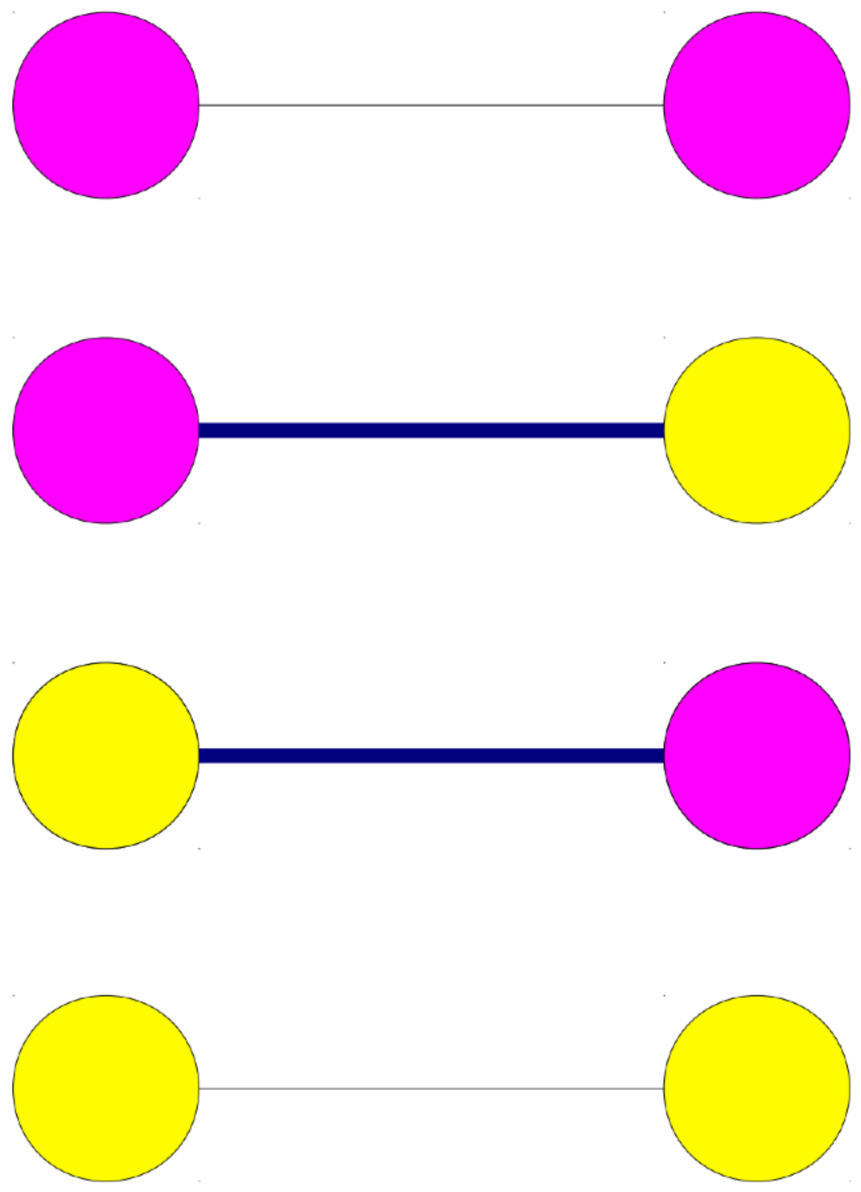
Analyzing the Max-Cut Algorithm



Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$
- $\text{Pr}[e \text{ crosses the cut}] = 1/2$
- $E[X] = \sum_e \frac{1}{2} = \frac{m}{2}$
- What is the maximum number of edges that can cross *any* cut?
    - $\text{OPT} \leq m$

Analyzing the Max-Cut Algorithm



Expected number of edges crossing the cut is then

- $E[X] = E[\sum_e C_e] = \sum_e E[C_e] = \sum_e \text{Pr}[e \text{ crosses the cut}]$
- Pr$[e$ crosses the cut$] = 1/2$
- $E[X] = \sum_e \frac{1}{2} = \frac{m}{2}$
- What is the maximum number of edges that can cross *any* cut?
    - OPT $\leq m$
- Thus, our randomized algorithm has
  an expected approximation ratio at least $1/2$, as it produces a cut
  of size at least $1/2$ of OPT in expectation

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
    - It is NP-hard to find the max-cut, but

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
  - It is NP-hard to find the max-cut, but
  - It is not hard at all (in expectation) to find a cut that is at least half the size of the max-cut!

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
  - It is NP-hard to find the max-cut, but
  - It is not hard at all (in expectation) to find a cut that is at least half the size of the max-cut!

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
  - It is NP-hard to find the max-cut, but
  - It is not hard at all (in expectation) to find a cut that is at least half the size of the max-cut!
- Can one do better than $1/2$?

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
    - It is NP-hard to find the max-cut, but
    - It is not hard at all (in expectation) to find a cut that is at least half the size of the max-cut!

- Can one do better than $1/2$?
    - Can get $\approx .878$ using extremely advanced techniques [Goemans, Williamson 95]

Expected Approximation Ratio

- Thus, our randomized algorithm, in expectation, has an approximation ratio $1/2$
- Monte-Carlo algorithm
- Running time: $O(n)$
- Takeaway:
  - It is NP-hard to find the max-cut, but
  - It is not hard at all (in expectation) to find a cut that is at least half the size of the max-cut!

- Can one do better than $1/2$?
  - Can get $\approx .878$ using extremely advanced techniques [Goemans, Williamson 95]
  - Might be optimal. Better than $.941$ is NP-hard

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - MIT course notes, 6.042/18.062J Mathematics for Computer Science April 26, 2005, Devadas and Lehman