PCP

Lecture 26
And Hardness of Approximation

Promise Problems

Promise Problems

- Decision problems, but with "don't cares"

Promise Problems

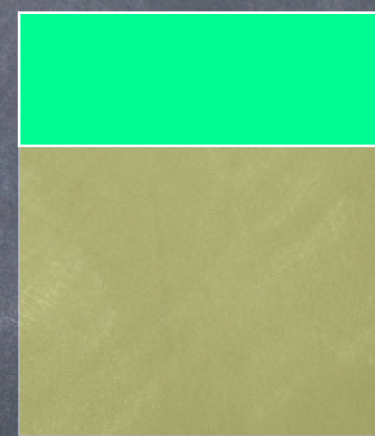
- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint

Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint

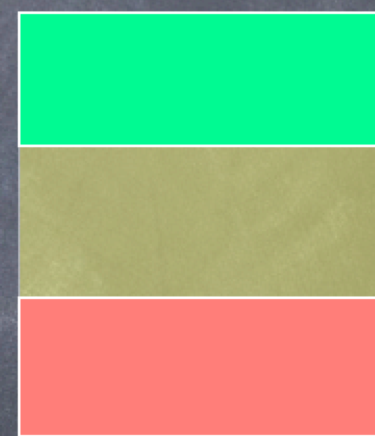
Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint



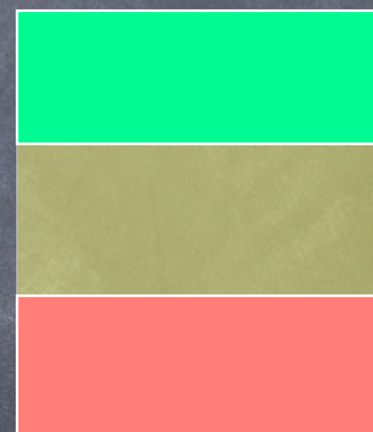
Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint



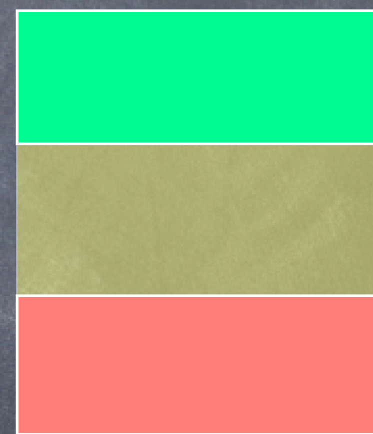
Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint
  - A TM is said to decide a promise problem if it correctly answers Yes or No for inputs from these sets



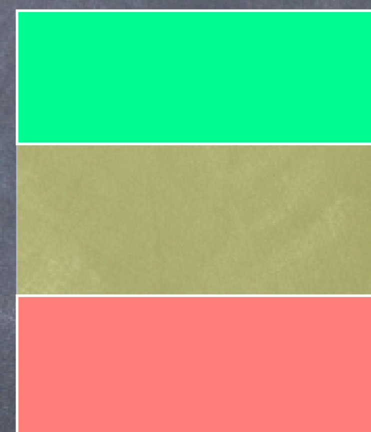
Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint
  - A TM is said to decide a promise problem if it correctly answers Yes or No for inputs from these sets
  - For inputs outside the two, don't care



Promise Problems

- Decision problems, but with "don't cares"
- Specified by a Yes set and a No set, disjoint
  - A TM is said to decide a promise problem if it correctly answers Yes or No for inputs from these sets
  - For inputs outside the two, don't care
    - We're "promised" that such inputs are not given



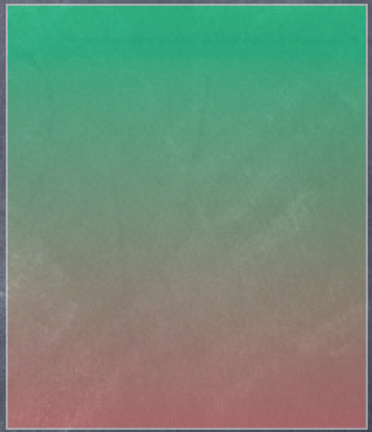
Gap Problems

Gap Problems

- Non-boolean functions (e.g. optimization problems)

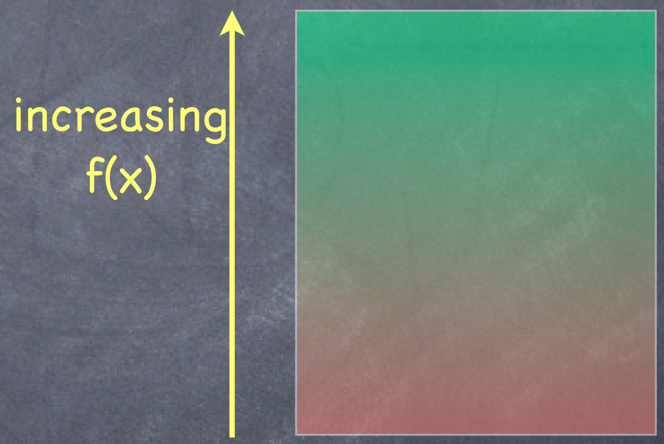
Gap Problems

- Non-boolean functions (e.g. optimization problems)



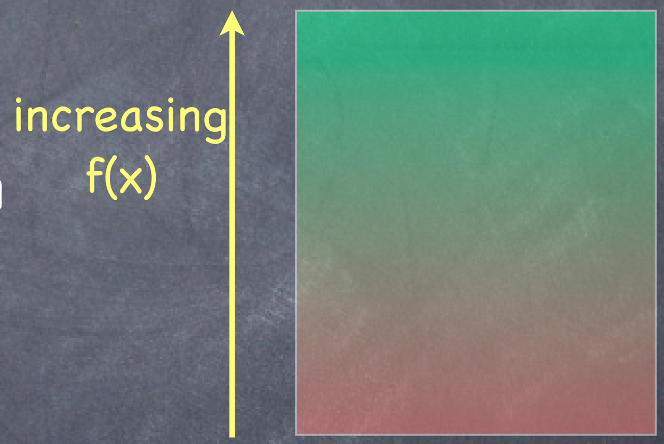
Gap Problems

- Non-boolean functions (e.g. optimization problems)



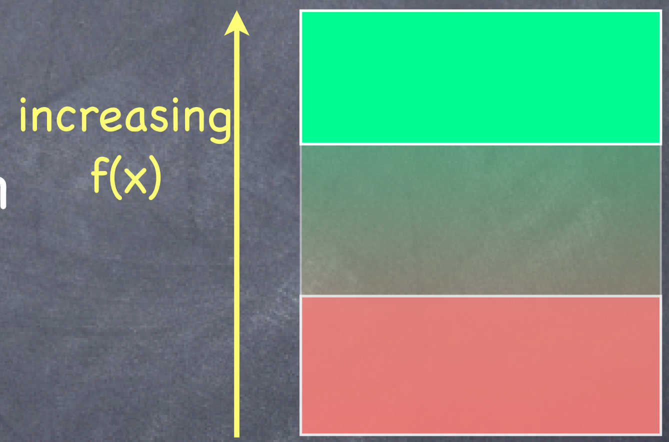
Gap Problems

- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value



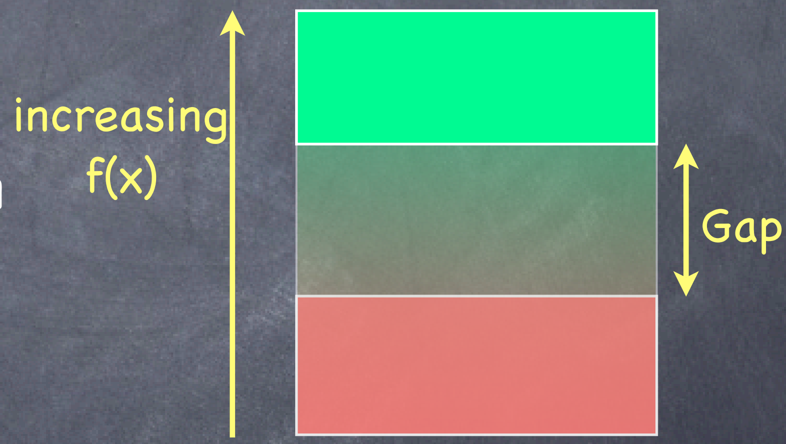
Gap Problems

- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value



Gap Problems

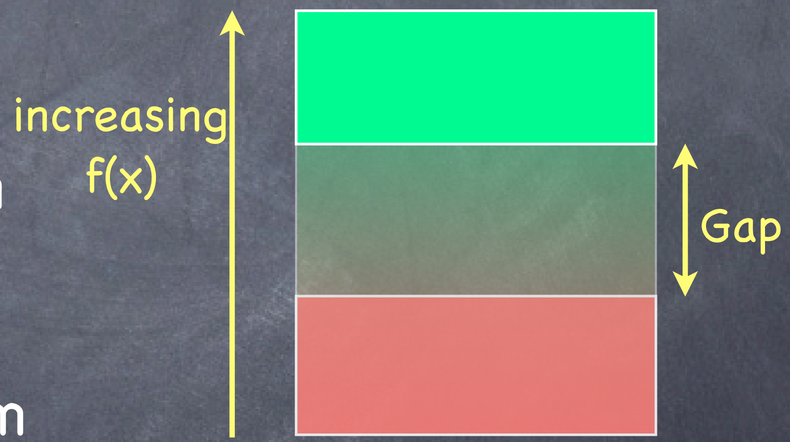
- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value



Gap Problems

- Non-boolean functions (e.g. optimization problems)

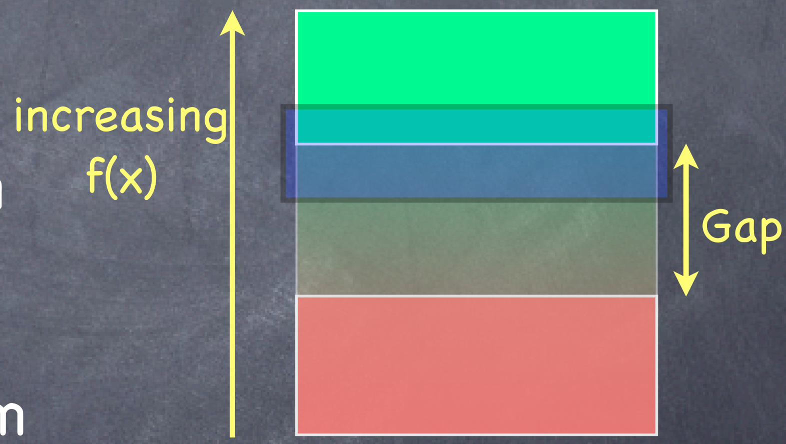
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value



- Can use an approximation algorithm for the function to solve the gap problem

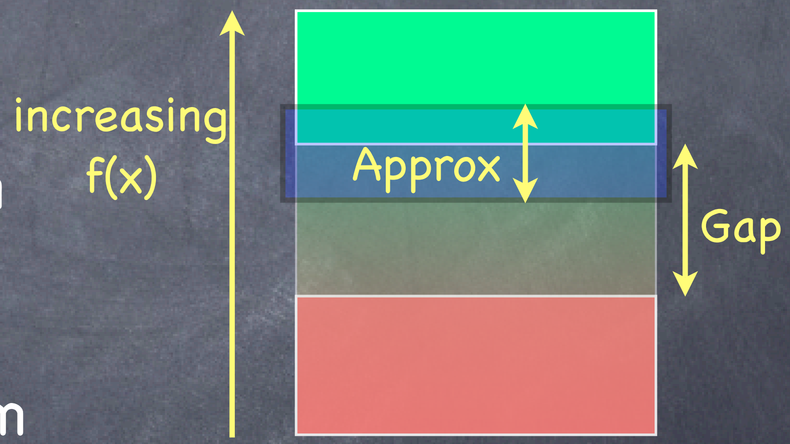
Gap Problems

- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value
- Can use an approximation algorithm for the function to solve the gap problem



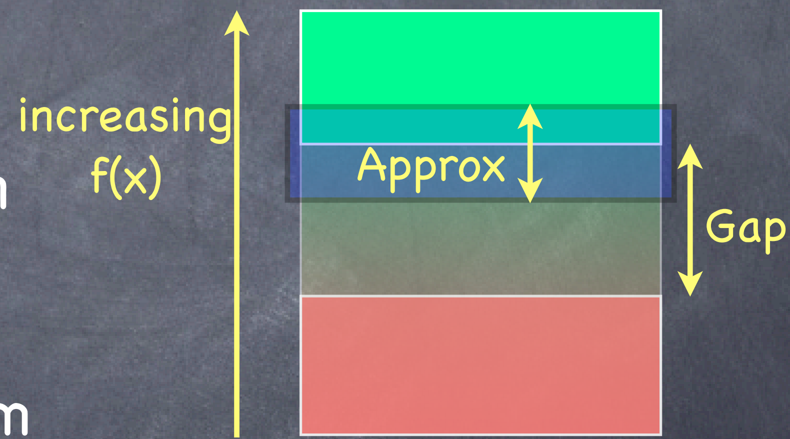
Gap Problems

- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value
- Can use an approximation algorithm for the function to solve the gap problem



Gap Problems

- Non-boolean functions (e.g. optimization problems)
- Gap problems: Promise problem in which Yes and No sets are separated by a gap in the function value
- Can use an approximation algorithm for the function to solve the gap problem
  - The more the gap the more loose the approximation can be



Certificates for a Gap problem

Certificates for a Gap problem

- A proof that the instance is a Yes instance

Certificates for a Gap problem

- A proof that the instance is a Yes instance
  - A probabilistically checkable proof (PCP): specified using the proof checking strategy

Certificates for a Gap problem

- A proof that the instance is a Yes instance
  - A probabilistically checkable proof (PCP): specified using the proof checking strategy
    - Completeness: If $x \in$ Yes, some proof accepted (with prob. 1)

Certificates for a Gap problem

- A proof that the instance is a Yes instance
  - A probabilistically checkable proof (PCP): specified using the proof checking strategy
    - Completeness: If $x \in$ Yes, some proof accepted (with prob. 1)
    - Soundness: If $x \in$ No, all proofs rejected with prob. > 1/2

Certificates for a Gap problem

- A proof that the instance is a Yes instance
  - A probabilistically checkable proof (PCP): specified using the proof checking strategy
    - Completeness: If $x \in$ Yes, some proof accepted (with prob. 1)
    - Soundness: If $x \in$ No, all proofs rejected with prob. $> 1/2$
- Parameters of interest: (r,q) where verifier tosses at most r coins and reads at most q bits

Certificates for a Gap problem

- A proof that the instance is a Yes instance
  - A probabilistically checkable proof (PCP): specified using the proof checking strategy
    - Completeness: If $x \in$ Yes, some proof accepted (with prob. 1)
    - Soundness: If $x \in$ No, all proofs rejected with prob. > 1/2
- Parameters of interest: (r,q) where verifier tosses at most r coins and reads at most q bits
  - Proof can be limited to be at most $q2^r$ bits long

PCP and CSP

PCP and CSP

- Constraint Satisfaction Problem (CSP)

PCP and CSP

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables

PCP and CSP

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables

Constraints: Arbitrary poly-time programs

PCP and CSP

Constraints: Arbitrary poly-time programs

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables
  - Yes instance: there exists an assignment of values to the variables such that all constraints are satisfied

PCP and CSP

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables
  - Yes instance: there exists an assignment of values to the variables such that all constraints are satisfied
  - No instance: for all assignments, less than half the constraints are satisfied

Constraints: Arbitrary poly-time programs

PCP and CSP

Constraints: Arbitrary poly-time programs

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables
  - Yes instance: there exists an assignment of values to the variables such that all constraints are satisfied
  - No instance: for all assignments, less than half the constraints are satisfied
  - (optimization problem: Max-CSPSat)

PCP and CSP

- Constraint Satisfaction Problem (CSP)
  - Instance specified by a set of "constraints" on R variables
  - Yes instance: there exists an assignment of values to the variables such that all constraints are satisfied
  - No instance: for all assignments, less than half the constraints are satisfied
  - (optimization problem: Max-CSPSat)
- A (gap) problem has a PCP iff can be reduced to CSP

Constraints: Arbitrary poly-time programs

PCP and CSP

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof
- Constraints are the verifier program with different random tapes: constraint is satisfied by the assignment if the verifier accepts the proof

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof
- Constraints are the verifier program with different random tapes: constraint is satisfied by the assignment if the verifier accepts the proof
  - Verifier accepts w/ prob. = 1 $\leftrightarrow$ All constraints satisfied

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof
- Constraints are the verifier program with different random tapes: constraint is satisfied by the assignment if the verifier accepts the proof
  - Verifier accepts w/ prob. = 1 $\leftrightarrow$ All constraints satisfied
  - Verifier accepts w/ prob. < 1/2 $\leftrightarrow$ Less than half satisfied

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof
- Constraints are the verifier program with different random tapes: constraint is satisfied by the assignment if the verifier accepts the proof
  - Verifier accepts w/ prob. = 1 $\leftrightarrow$ All constraints satisfied
  - Verifier accepts w/ prob. < 1/2 $\leftrightarrow$ Less than half satisfied
- qCSP with m constraints: each constraint involves q variables

PCP and CSP

- A (gap) problem has a PCP iff can be reduced to CSP
- Variables are the bits of the proofs: assignment is a proof
- Constraints are the verifier program with different random tapes: constraint is satisfied by the assignment if the verifier accepts the proof
  - Verifier accepts w/ prob. = 1 $\leftrightarrow$ All constraints satisfied
  - Verifier accepts w/ prob. < 1/2 $\leftrightarrow$ Less than half satisfied
- qCSP with m constraints: each constraint involves q variables
  - PCP(log m,q): q-query (non-adaptive) verifier, tosses at most log m coins

Decision Problem to Gap Problem

Decision Problem to Gap Problem

- Reducing a decision problem (language) L to a gap problem G

Decision Problem to Gap Problem

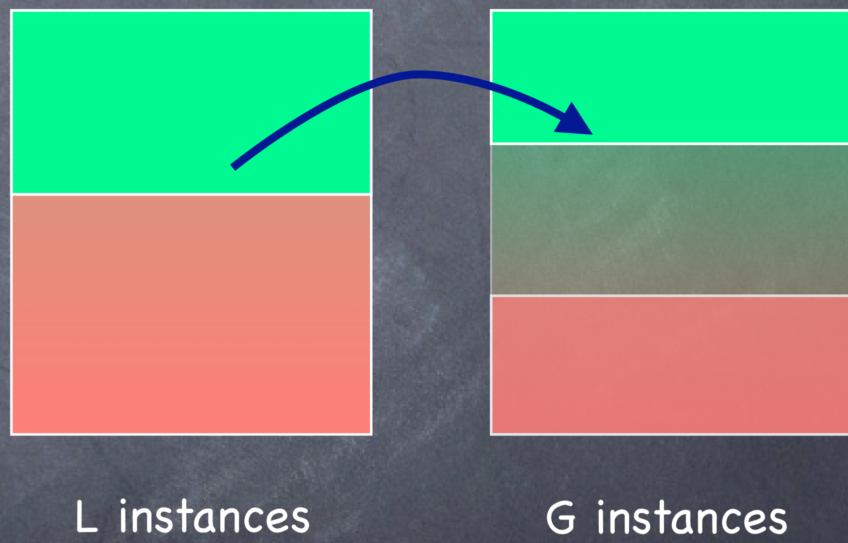
- Reducing a decision problem (language) L to a gap problem G
  - "Separating" Yes and No

Decision Problem to Gap Problem

- Reducing a decision problem (language) L to a gap problem G
  - "Separating" Yes and No
  - If L is hard, and can do the reduction efficiently, then approximating the function underlying G should be hard

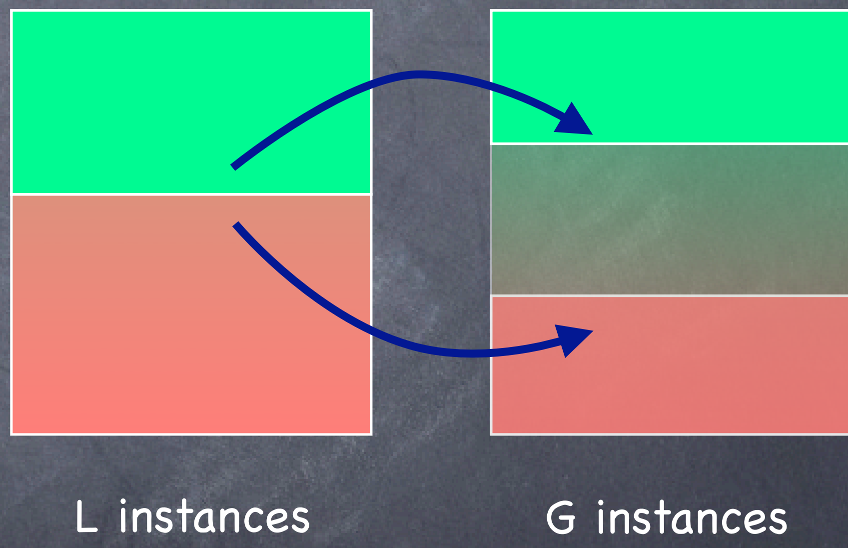
Decision Problem to Gap Problem

- Reducing a decision problem (language) L to a gap problem G
  - "Separating" Yes and No
  - If L is hard, and can do the reduction efficiently, then approximating the function underlying G should be hard



Decision Problem to Gap Problem

- Reducing a decision problem (language) L to a gap problem G
  - "Separating" Yes and No
  - If L is hard, and can do the reduction efficiently, then approximating the function underlying G should be hard



PCP Theorem

PCP Theorem

- Can reduce any NP language to qCSP

PCP Theorem

- Can reduce any NP language to qCSP

A gap problem, with gap=1/2

PCP Theorem

- Can reduce any NP language to qCSP
  - With m = poly(n) constraints and q = O(1)

A gap problem, with gap=1/2

PCP Theorem

- Can reduce any NP language to qCSP
  - With m = poly(n) constraints and q = O(1)

A gap problem, with gap=1/2

- Since qCSP has a PCP (with r=log m, and q=q), any NP language has a PCP

PCP Theorem

- Can reduce any NP language to qCSP

A gap problem, with gap=1/2

  - With m = poly(n) constraints and q = O(1)
- Since qCSP has a PCP (with r=log m, and q=q), any NP language has a PCP
  - NP ⊆ PCP(log n, 1)

PCP Theorem

- Can reduce any NP language to qCSP

  A gap problem, with gap=1/2

  - With m = poly(n) constraints and q = O(1)

- Since qCSP has a PCP (with r=log m, and q=q), any NP language has a PCP

  - NP ⊆ PCP(log n, 1)

  PCP(r,q): Class of languages with r-coin, q-query PCP verifiers

PCP Theorem

- Can reduce any NP language to qCSP

  A gap problem, with gap=1/2

  - With m = poly(n) constraints and q = O(1)

- Since qCSP has a PCP (with r=log m, and q=q), any NP language has a PCP

  - NP ⊆ PCP(log n, 1)

    PCP(r,q): Class of languages with r-coin, q-query PCP verifiers

- Note: PCP(log n, *) ⊆ NP

PCP Theorem

- Can reduce any NP language to qCSP

A gap problem, with gap=1/2

  - With m = poly(n) constraints and q = O(1)

- Since qCSP has a PCP (with r=log m, and q=q), any NP language has a PCP

  - NP ⊆ PCP(log n, 1)

PCP(r,q): Class of languages with r-coin, q-query PCP verifiers

- Note: PCP(log n, *) ⊆ NP

  - So, NP = PCP(log n, 1)

Hardness of Approximation

Hardness of Approximation

- By PCP theorem, Max-qCSPSat is hard to approximate within a factor of 1/2

Hardness of Approximation

- By PCP theorem, Max-qCSPSat is hard to approximate within a factor of 1/2
- How about Max-3SAT? Max-CLIQUE? Other NP-hard functions?

Hardness of Approximation

- By PCP theorem, Max-qCSPSat is hard to approximate within a factor of 1/2
- How about Max-3SAT? Max-CLIQUE? Other NP-hard functions?
  - Reduce Max-qCSPSat to these problems

Hardness of Approximation

- By PCP theorem, Max-qCSPSat is hard to approximate within a factor of 1/2
- How about Max-3SAT? Max-CLIQUE? Other NP-hard functions?
  - Reduce Max-qCSPSat to these problems
  - Such that approximation for them imply approximation for Max-qCSPSat

Gap-preserving Reductions

Gap-preserving Reductions

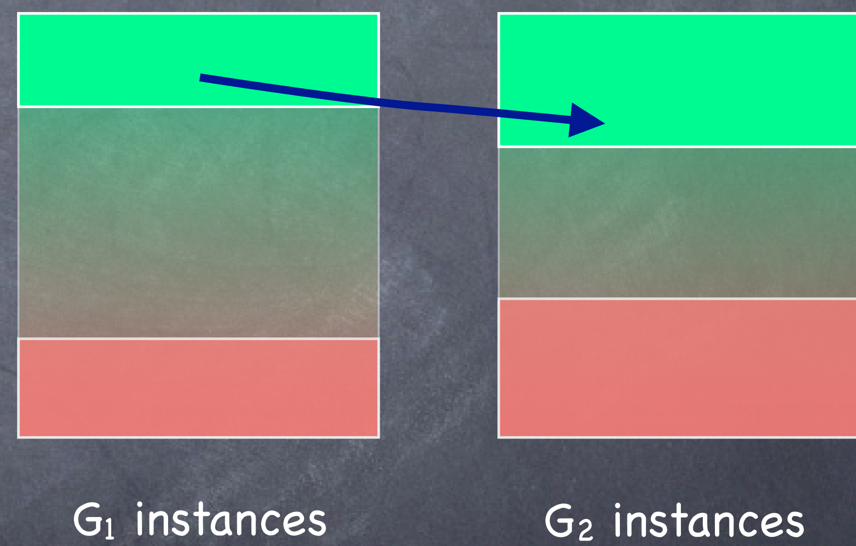
- From gap problem G_1 to G_2

Gap-preserving Reductions

- From gap problem G_1 to G_2

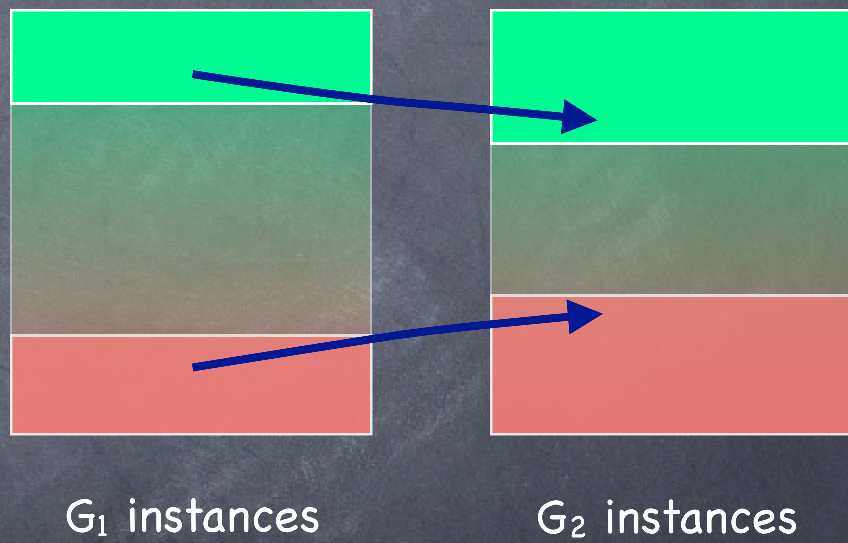
Gap-preserving Reductions

- From gap problem G_1 to G_2



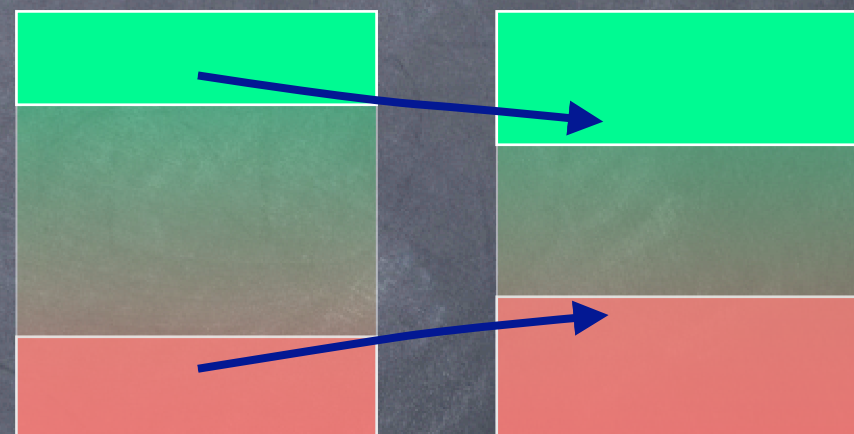
Gap-preserving Reductions

- From gap problem G_1 to G_2



Gap-preserving Reductions

- From gap problem G_1 to G_2
  - If G_1 is hard to solve and reduction is efficient, then G_2 is hard to solve

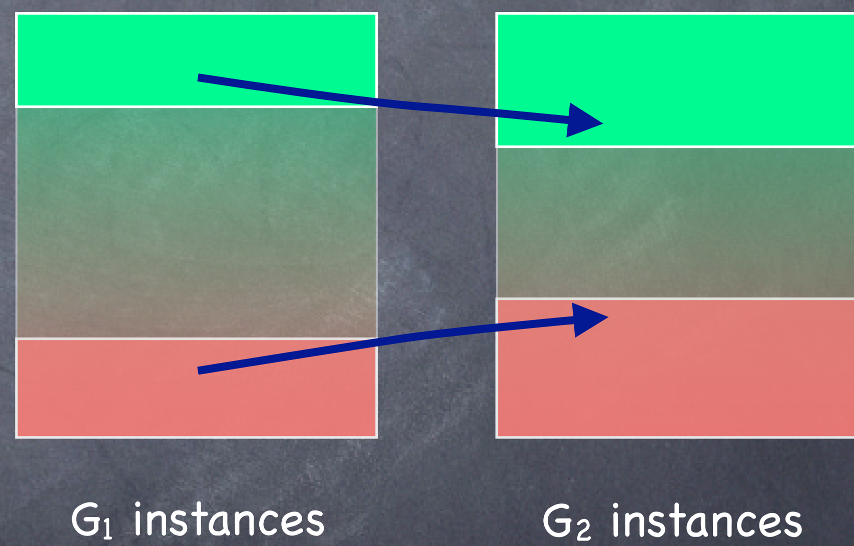


G_1 instances

G_2 instances

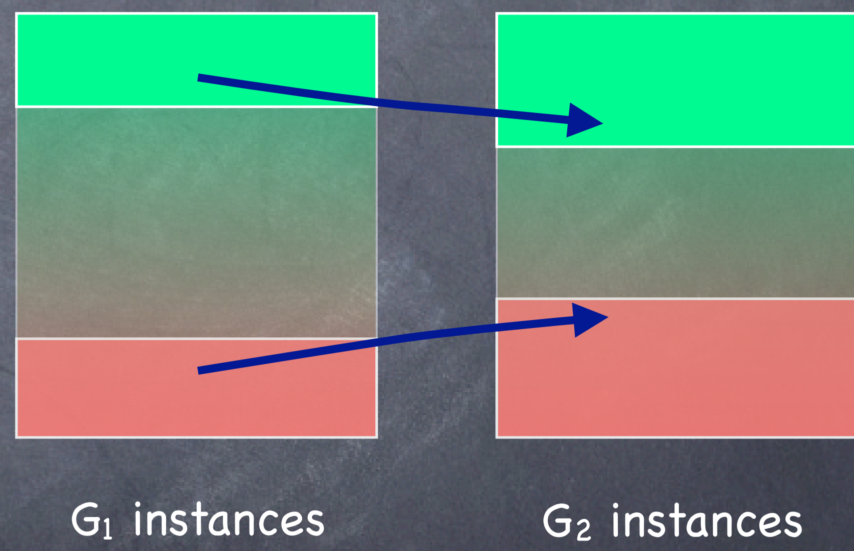
Gap-preserving Reductions

- From gap problem G_1 to G_2
  - If G_1 is hard to solve and reduction is efficient, then G_2 is hard to solve
    - Then function underlying G_2 is hard to approximate (within a factor of its gap)

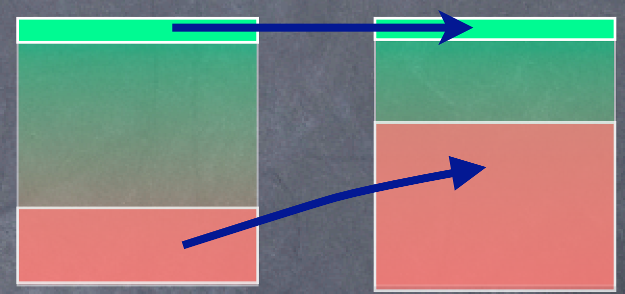


Gap-preserving Reductions

- From gap problem G_1 to G_2
  - If G_1 is hard to solve and reduction is efficient, then G_2 is hard to solve
    - Then function underlying G_2 is hard to approximate (within a factor of its gap)
    - The bigger the gap in G_2 the larger the approximation factor shown hard

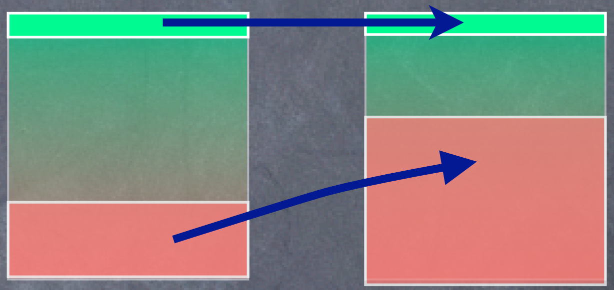


Max-qCSP to Max-3SAT



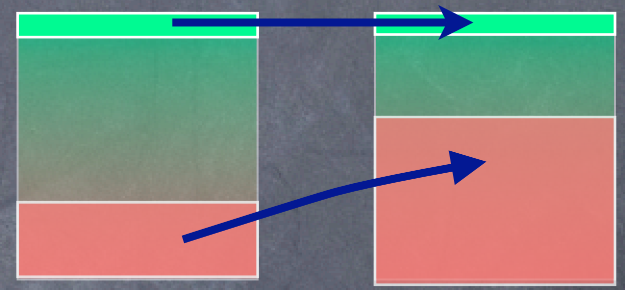
Max-qCSP to Max-3SAT

- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)



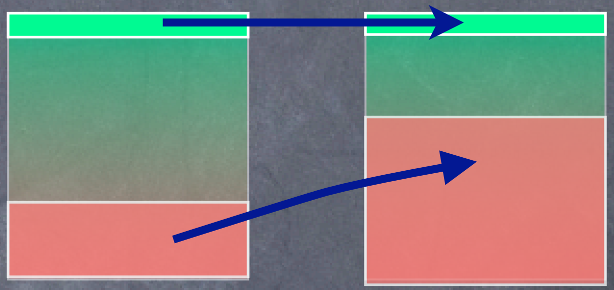
Max-qCSP to Max-3SAT

- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses



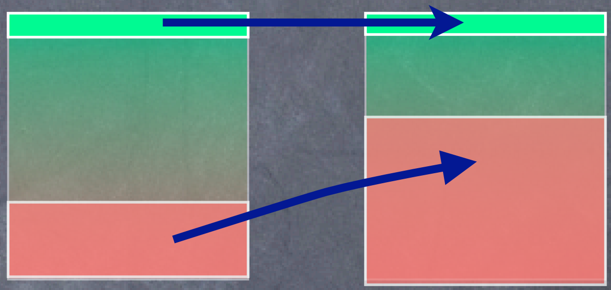
Max-qCSP to Max-3SAT

- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses
- Collect all clauses from all constraints



Max-qCSP to Max-3SAT

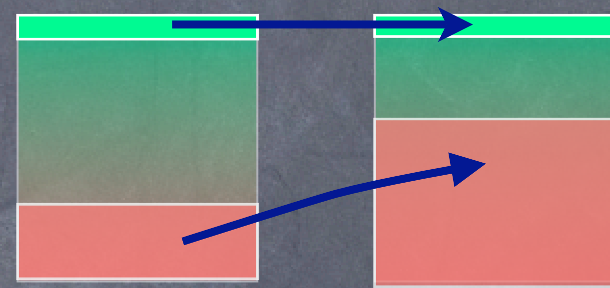
- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses
- Collect all clauses from all constraints



- So far gap is preserved up to a factor of $1/2^q$

Max-qCSP to Max-3SAT

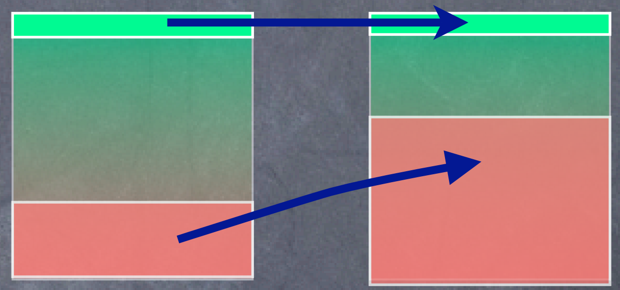
- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses



- Collect all clauses from all constraints
- So far gap is preserved up to a factor of $1/2^q$
- Now turn each q-clause into a collection of 3-clauses

Max-qCSP to Max-3SAT

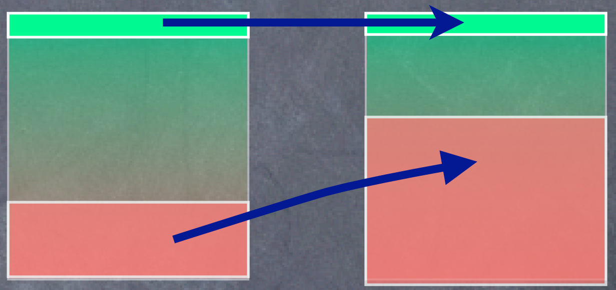
- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses



- Collect all clauses from all constraints
- So far gap is preserved up to a factor of $1/2^q$
- Now turn each q-clause into a collection of 3-clauses
  - Adding at most q auxiliary var.s to get at most q 3-clauses

Max-qCSP to Max-3SAT

- Write each constraint as an exponential sized CNF (AND-OR) formula, of clauses with q vars (q-clauses)
  - At most 2^q q-clauses



- Collect all clauses from all constraints
- So far gap is preserved up to a factor of $1/2^q$
- Now turn each q-clause into a collection of 3-clauses
  - Adding at most q auxiliary var.s to get at most q 3-clauses
- Gap preserved up to a factor of $1/(q2^q)$

Max-3SAT to Max-CLIQUE

Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE:
  Clauses → Graph

Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph

$(x \vee \neg y \vee \neg z)$

$(w \vee y)$

$(w \vee x \vee \neg z)$

Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)

$(x \lor \neg y \lor \neg z)$

$(w \lor y)$

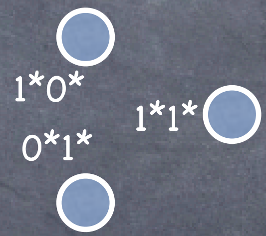
$(w \lor x \lor \neg z)$

Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)

$(x \lor \neg y \lor \neg z)$

$(w \lor y)$



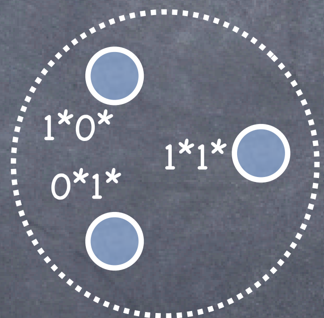
$(w \lor x \lor \neg z)$

Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
    - vertices: each clause's sat assignments (for its variables)

$(x \lor \neg y \lor \neg z)$

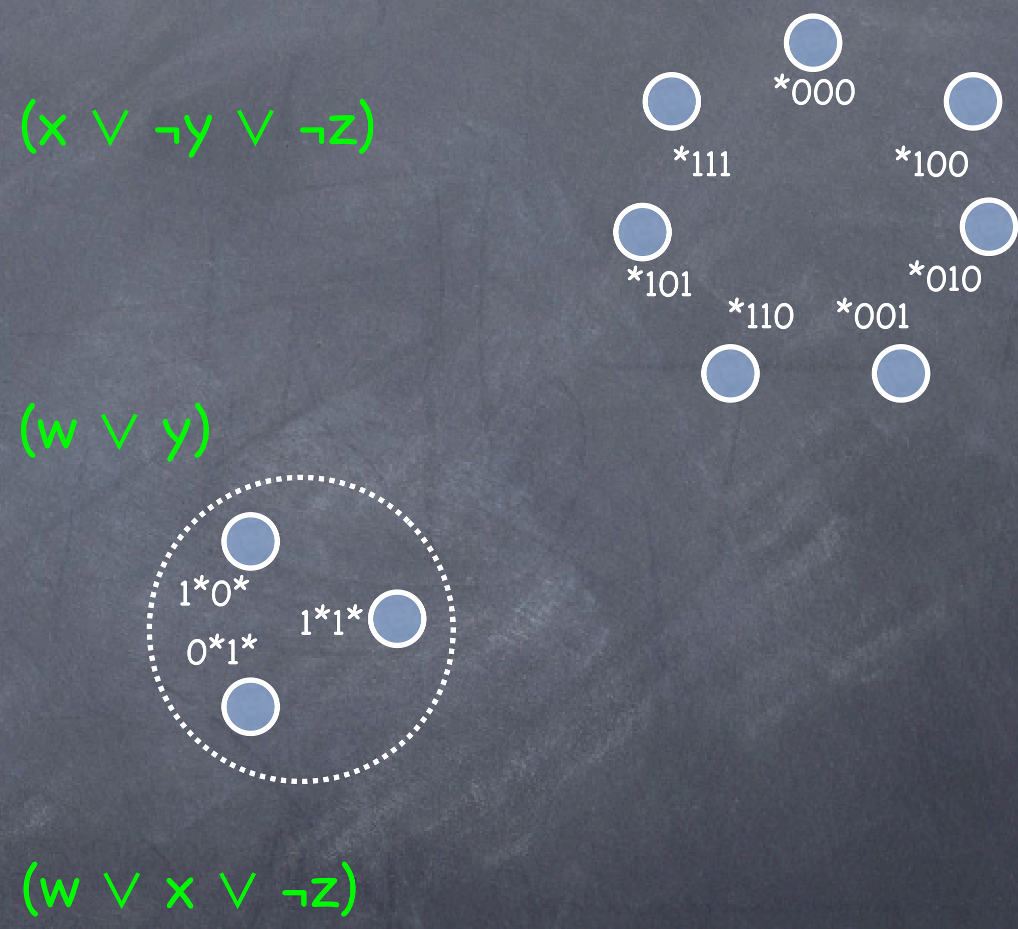
$(w \lor y)$



$(w \lor x \lor \neg z)$

Max-3SAT to Max-CLIQUE

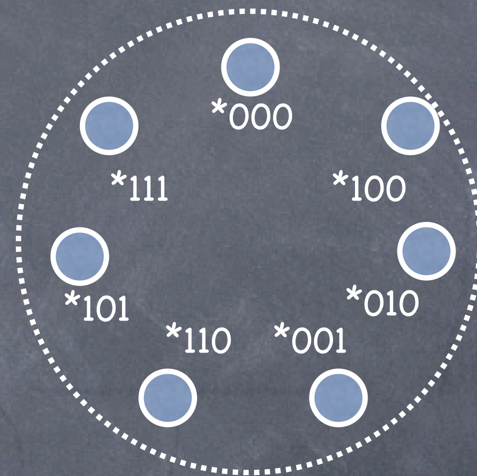
- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)



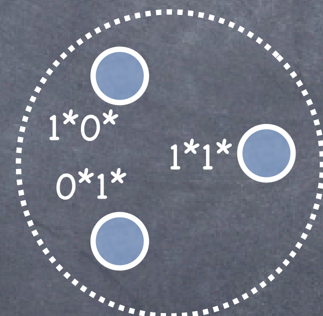
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)

$(x \vee \neg y \vee \neg z)$



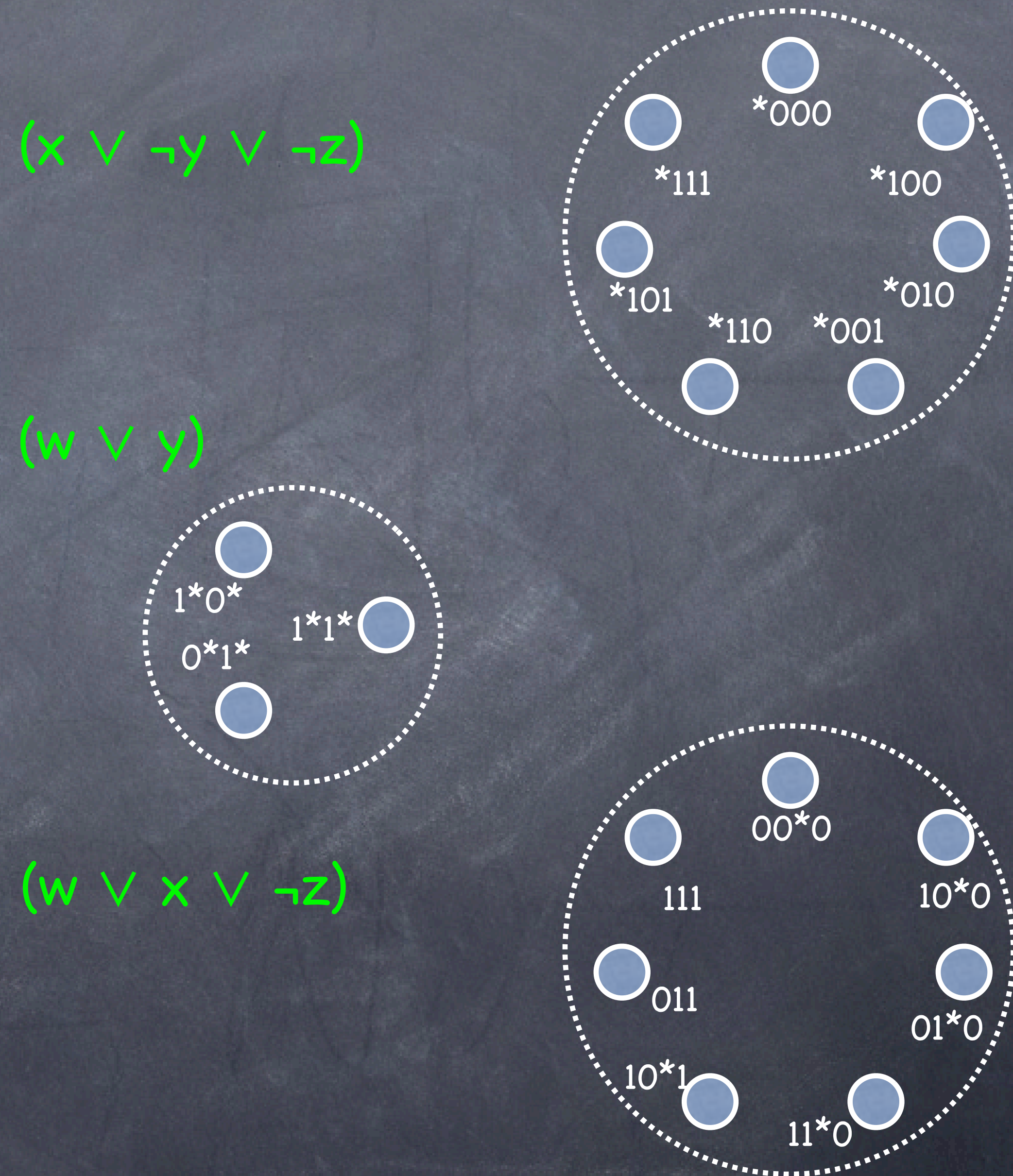
$(w \vee y)$



$(w \vee x \vee \neg z)$

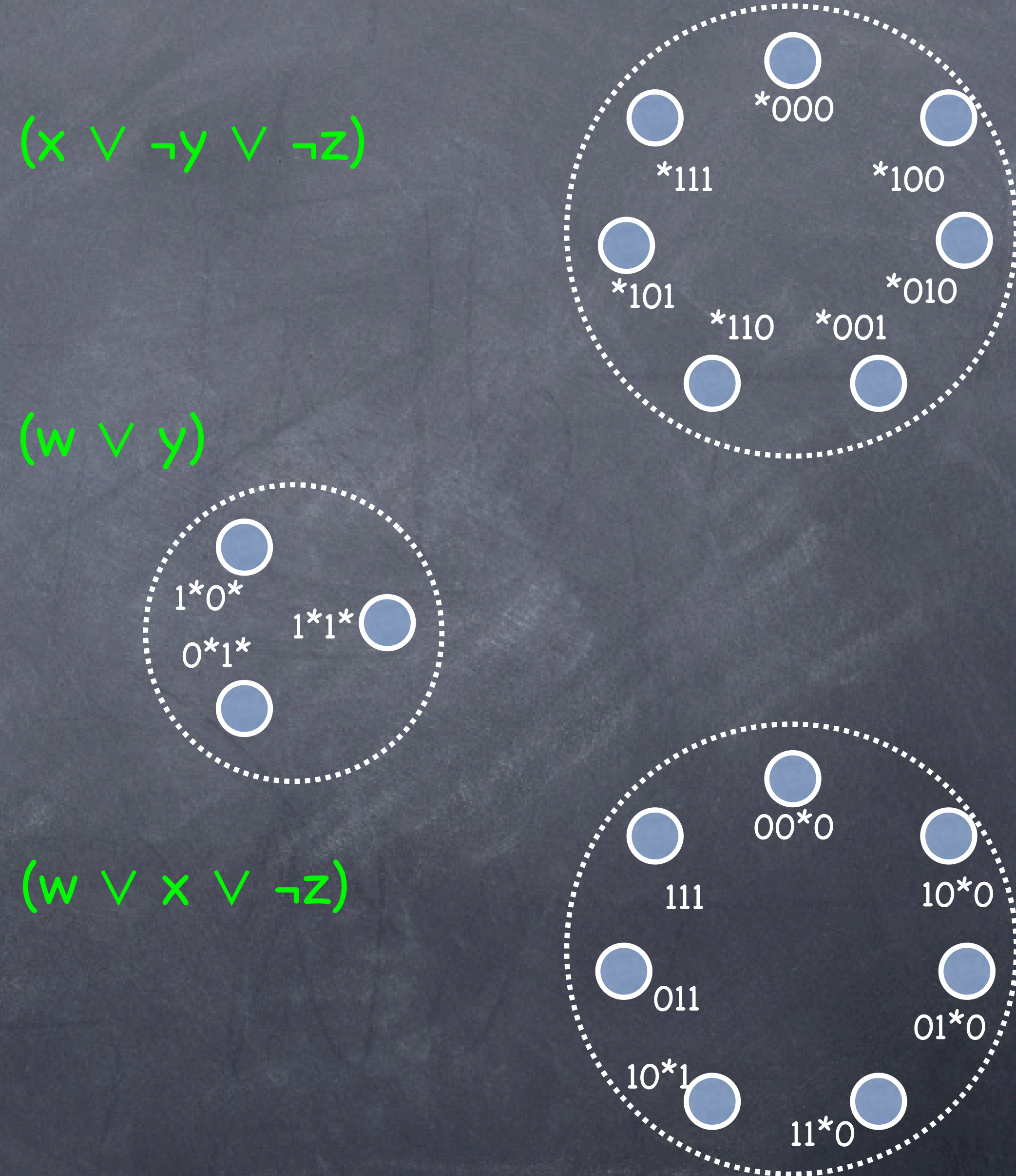
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)



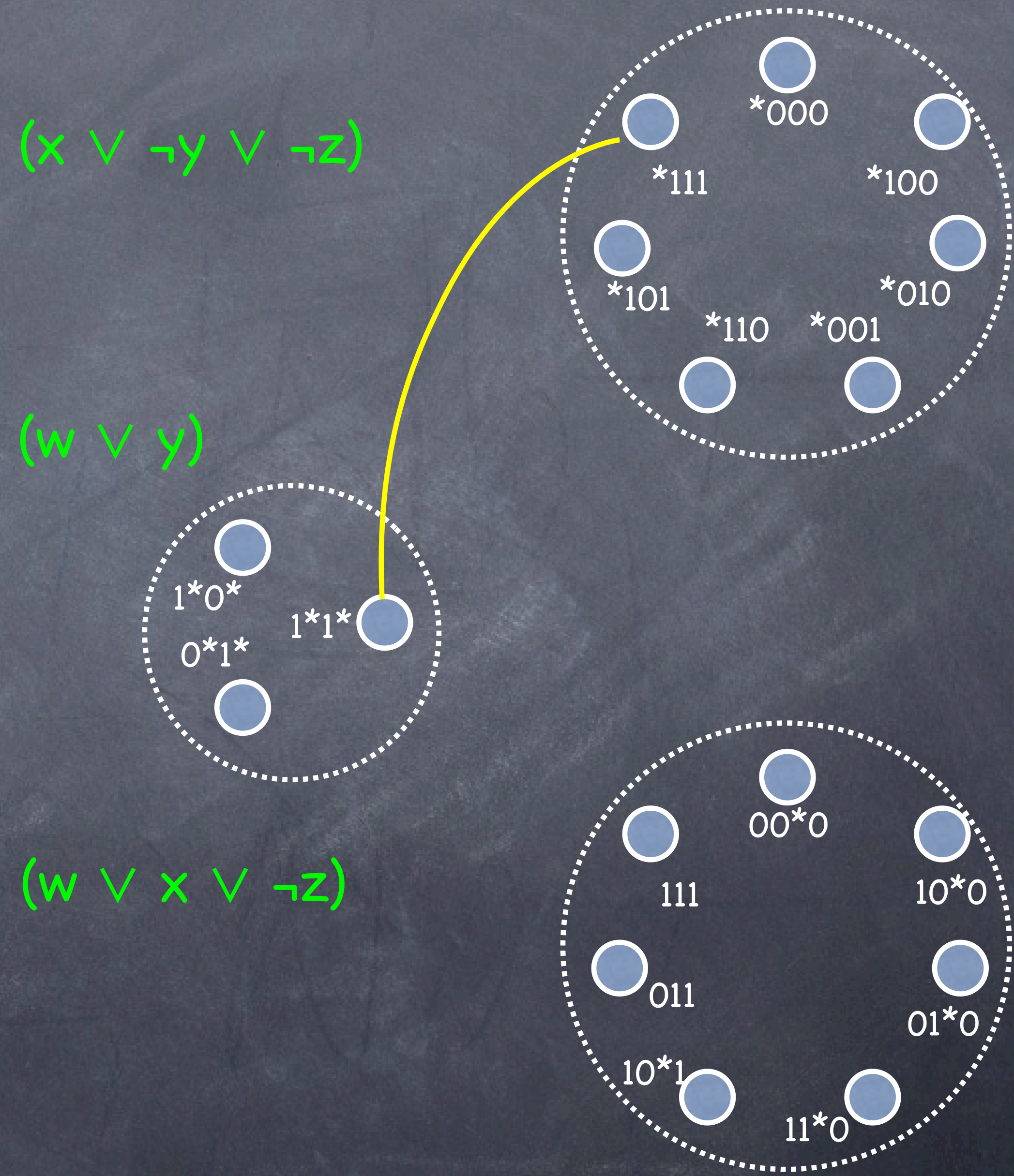
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



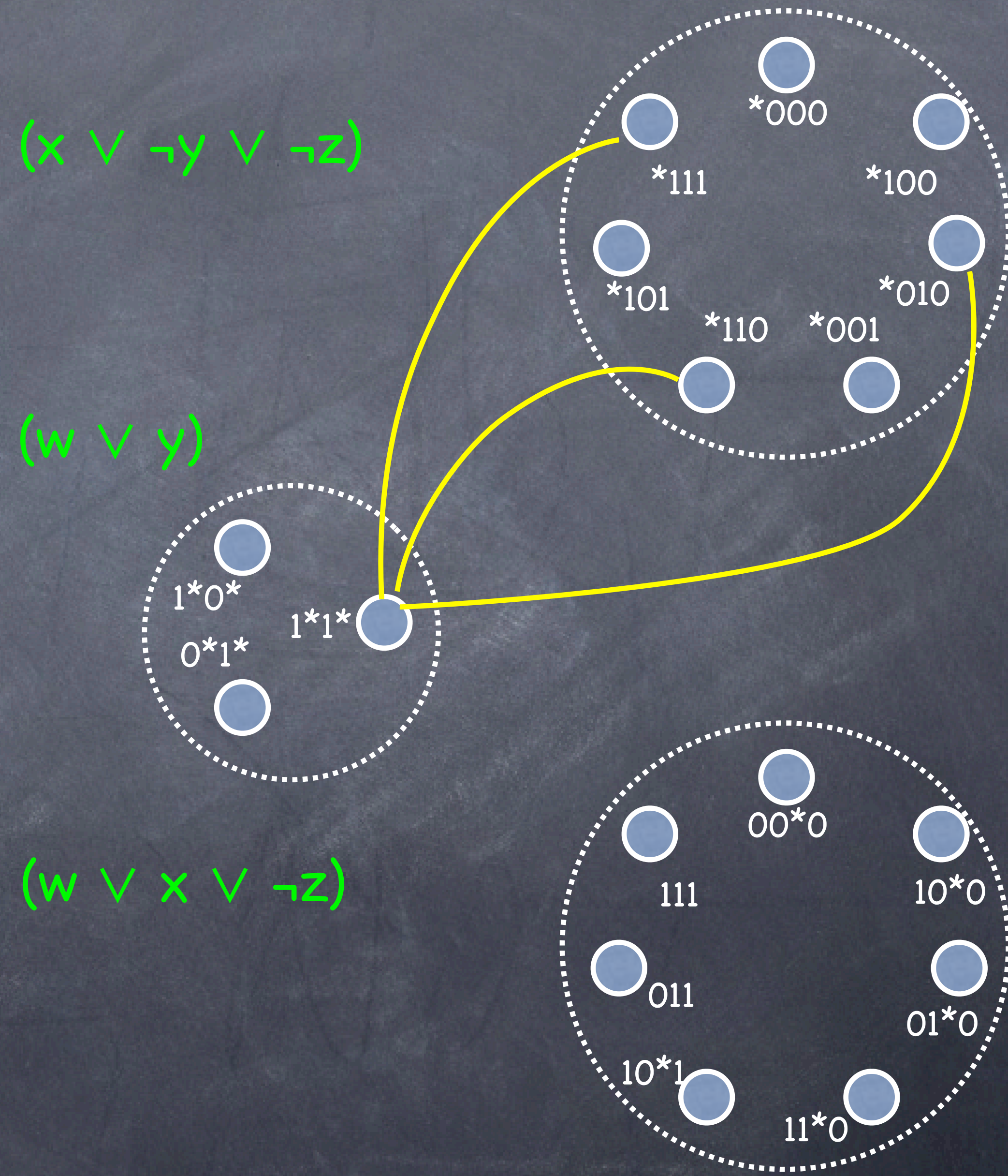
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



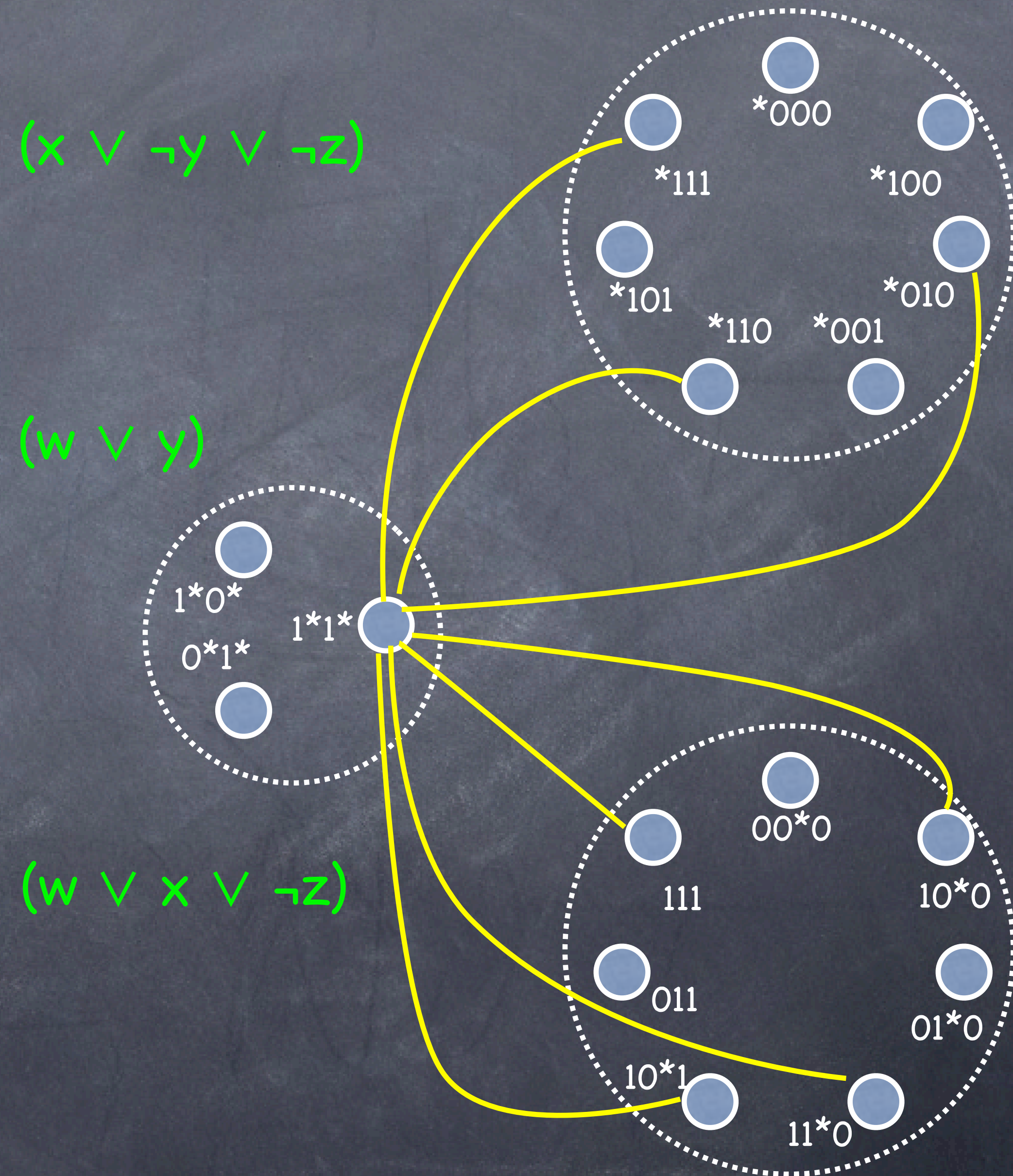
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



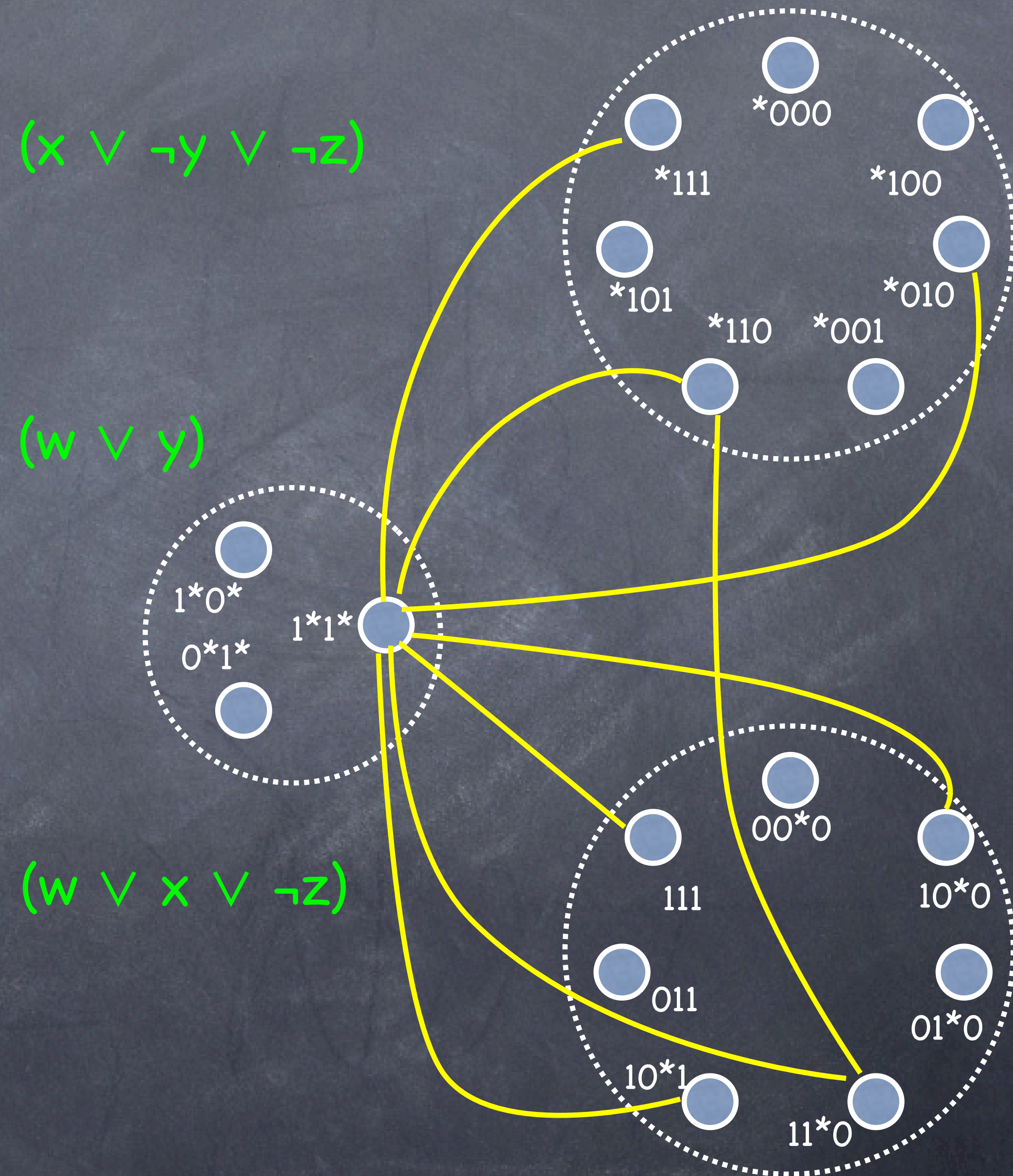
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



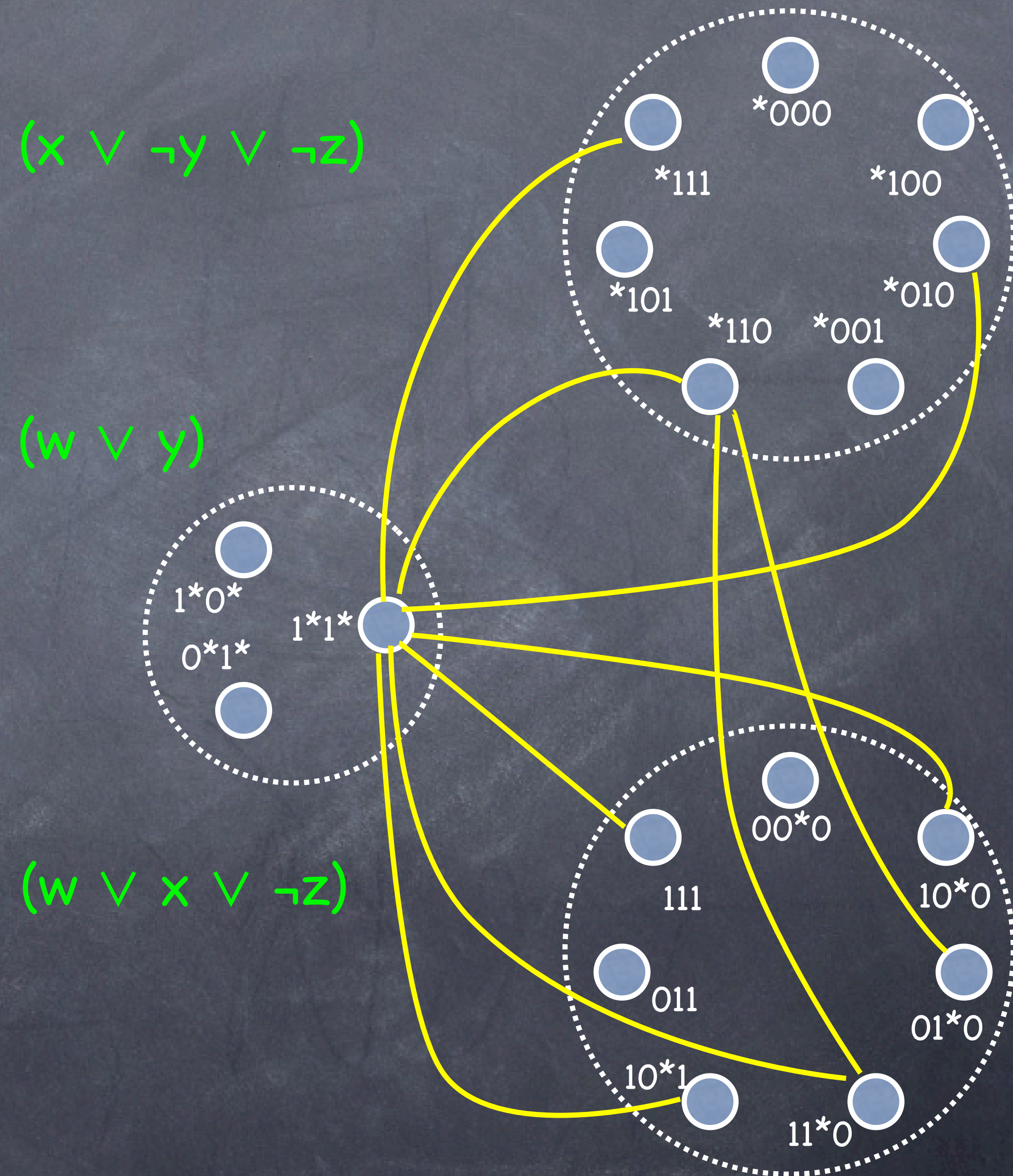
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



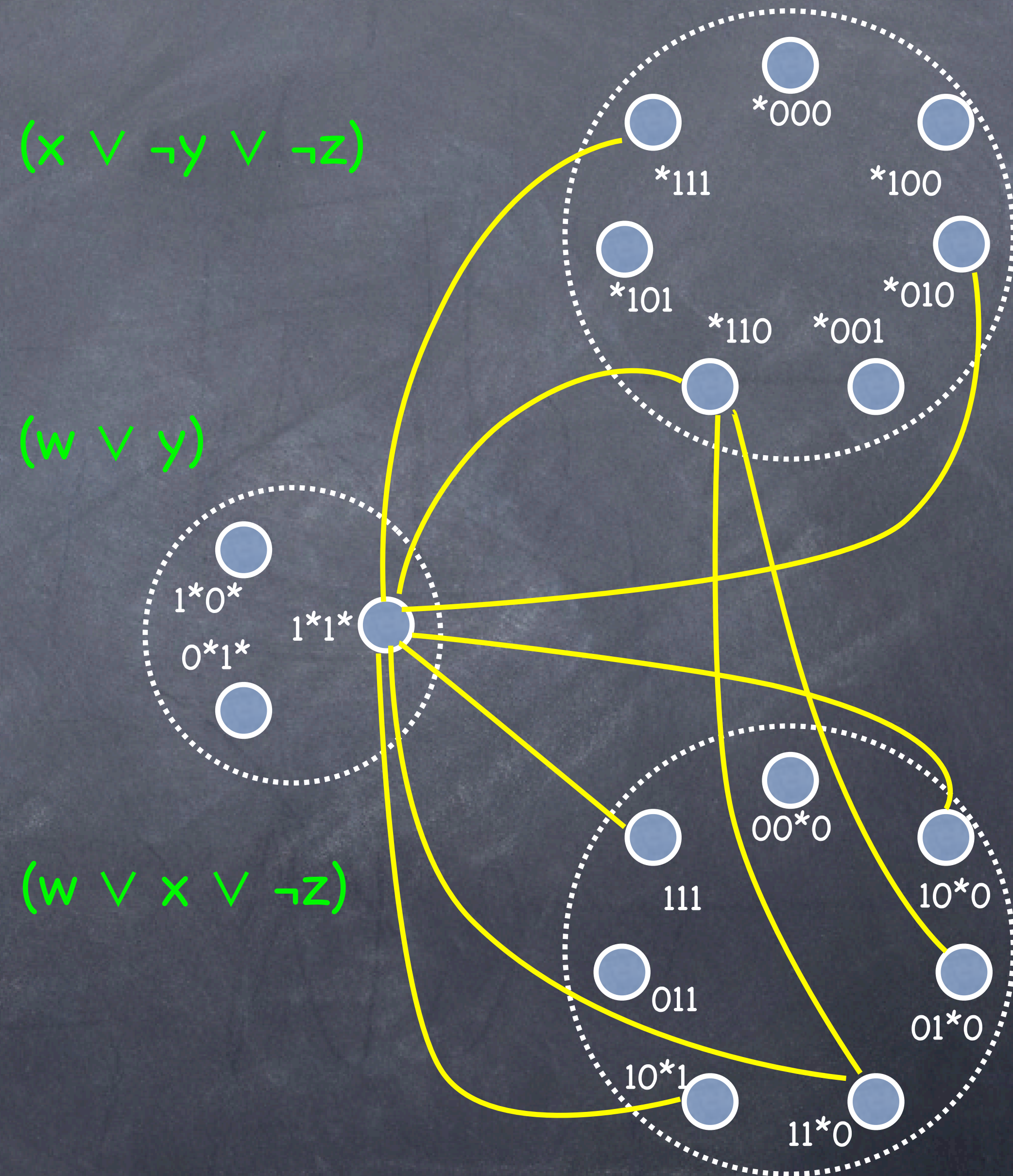
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments



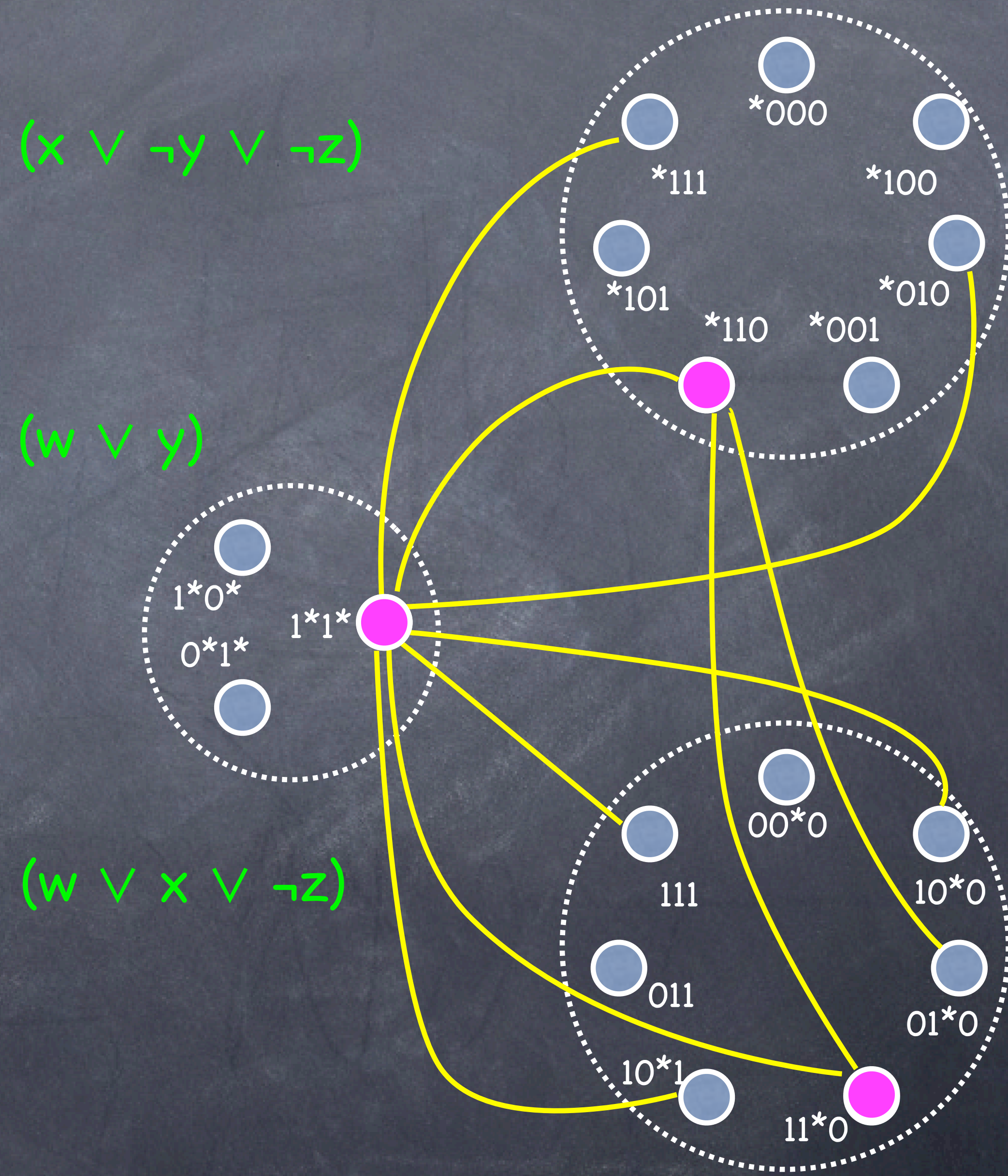
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments
  - k-clique iff k clauses satisfiable



Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments
  - k-clique iff k clauses satisfiable



Max-3SAT to Max-CLIQUE

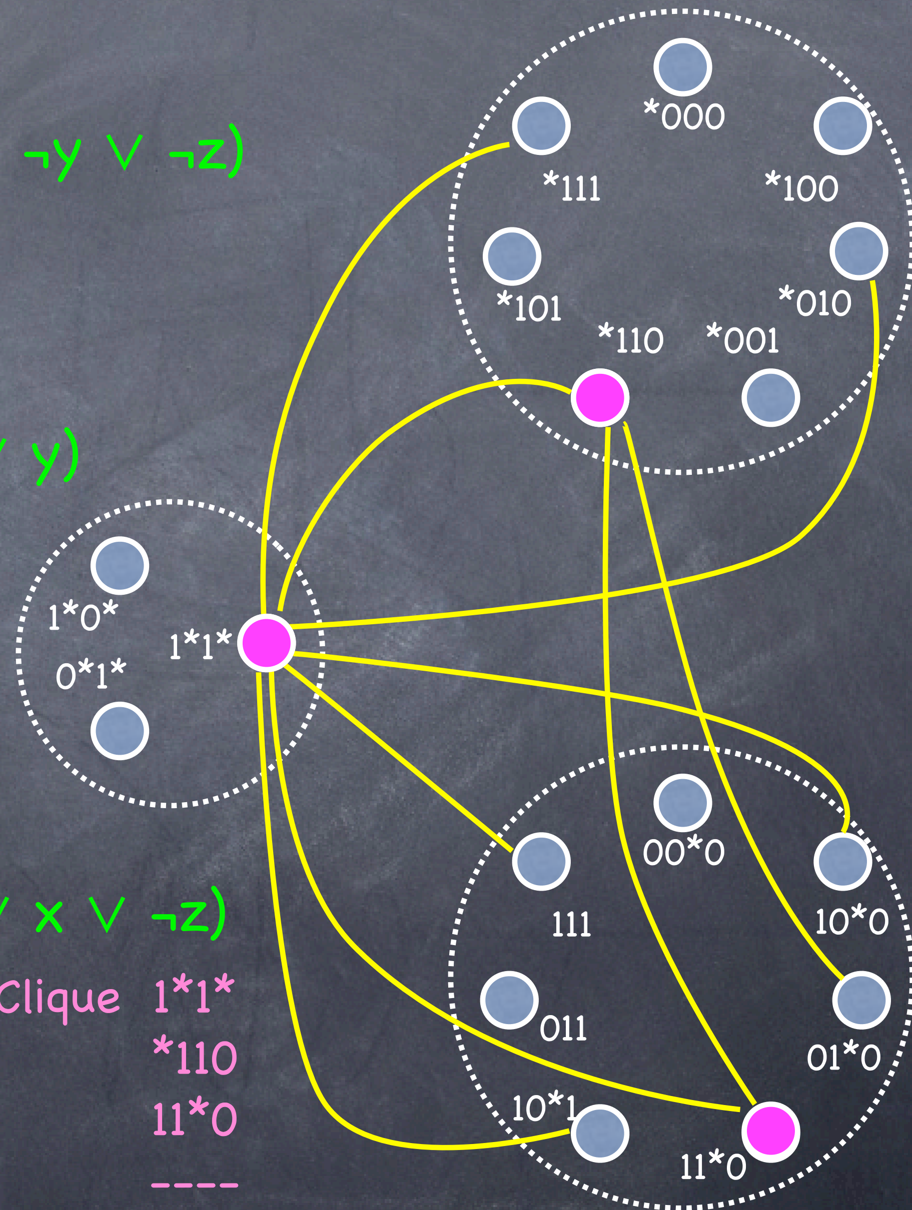
- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments
  - k-clique iff k clauses satisfiable

(x ∨ ¬y ∨ ¬z)

(w ∨ y)

(w ∨ x ∨ ¬z)

3-Clique 1*1*
*110
11*0
----
sat assignment 1110



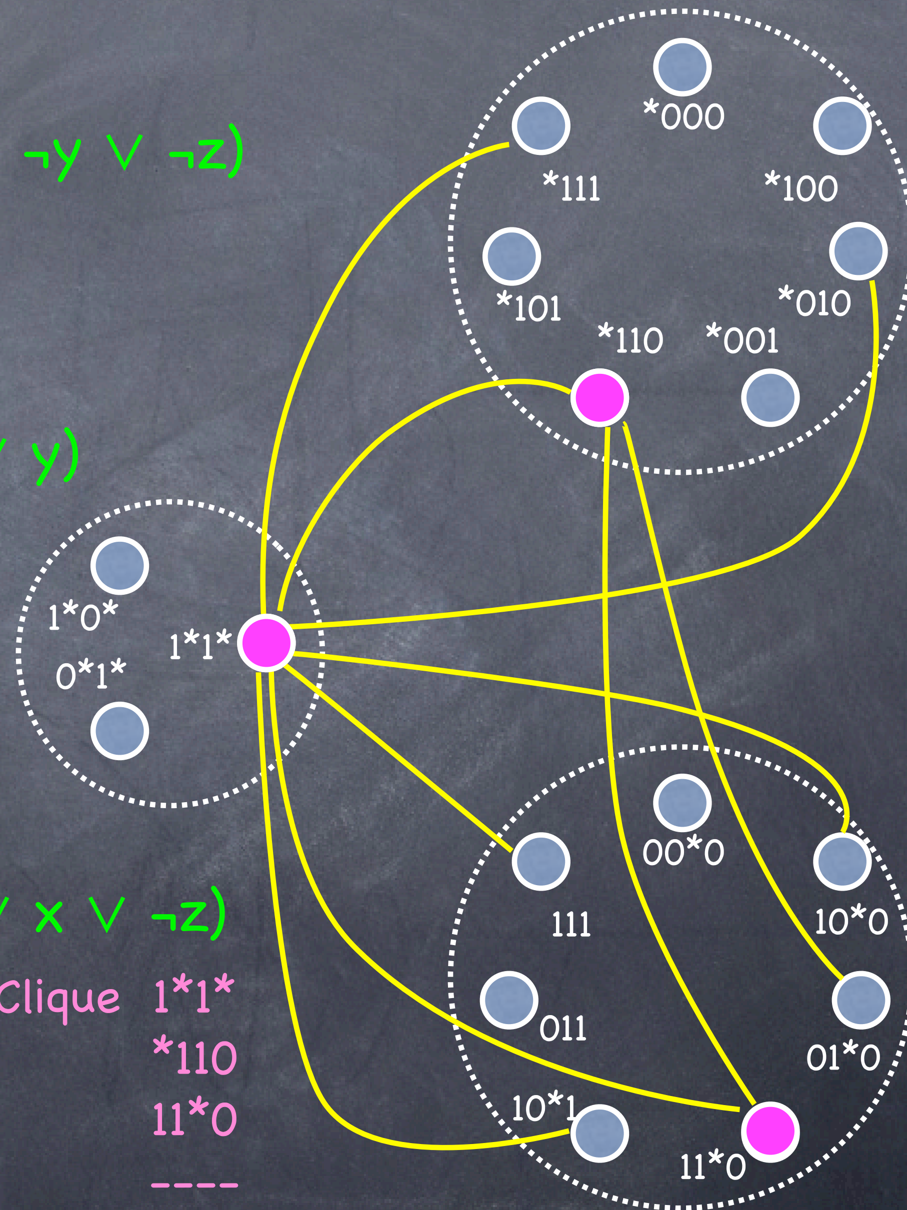
Max-3SAT to Max-CLIQUE

- Recall 3SAT to CLIQUE: Clauses → Graph
  - vertices: each clause's sat assignments (for its variables)
  - edges between consistent assignments
  - k-clique iff k clauses satisfiable
  - Gap preserved

(x ∨ ¬y ∨ ¬z)

(w ∨ y)

(w ∨ x ∨ ¬z)



3-Clique 1*1*
*110
11*0
----
sat assignment 1110

Proving the PCP Theorem

Proving the PCP Theorem

- Very involved: see textbook

Proving the PCP Theorem

- Very involved: see textbook
- A flavor:

Proving the PCP Theorem

- Very involved: see textbook
- A flavor:
  - Recall: to give a PCP system for 3SAT

Proving the PCP Theorem

- Very involved: see textbook
- A flavor:
  - Recall: to give a PCP system for 3SAT
  - i.e. need to check if all clauses satisfied by the assignment implicit in the proof

Proving the PCP Theorem

- Very involved: see textbook
- A flavor:
  - Recall: to give a PCP system for 3SAT
  - i.e. need to check if all clauses satisfied by the assignment implicit in the proof
    - Checking a random clause is no good (though it takes only 3 queries) as almost all clauses might be satisfied

Proving the PCP Theorem

- Very involved: see textbook
- A flavor:
  - Recall: to give a PCP system for 3SAT
  - i.e. need to check if all clauses satisfied by the assignment implicit in the proof
    - Checking a random clause is no good (though it takes only 3 queries) as almost all clauses might be satisfied
    - Need to check if any 1 in an implicit bit vector: checking a random position is no good

Proving the PCP Theorem

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good
  - Require a "robust" encoding to be given

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good
  - Require a “robust” encoding to be given
  - If even one 1, it becomes easy to detect

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good
  - Require a "robust" encoding to be given
  - If even one 1, it becomes easy to detect
    - e.g. Walsh-Hadamard code: consider n-bit vector x as a function $f_x(y) = \langle x,y \rangle$. Encoding is the truth-table

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good
  - Require a "robust" encoding to be given
  - If even one 1, it becomes easy to detect
    - e.g. Walsh-Hadamard code: consider n-bit vector x as a function $f_x(y) = \langle x,y \rangle$. Encoding is the truth-table
      - If one or more 1, then half 1s and half 0s. Else all 0s.

Proving the PCP Theorem

- Need to check if any 1 in an implicit bit vector: checking a random position is no good
  - Require a "robust" encoding to be given
  - If even one 1, it becomes easy to detect
    - e.g. Walsh-Hadamard code: consider n-bit vector x as a function $f_x(y) = \langle x,y \rangle$. Encoding is the truth-table
      - If one or more 1, then half 1s and half 0s. Else all 0s.
  - Need to check that the encoded vector is the evaluation of the clauses on an assignment, and that encoding is valid

Linearity Test

Linearity Test

- Is a function table provided close to being linear?

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors
    - Dot product with another boolean vector

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors
    - Dot product with another boolean vector
    - A function in the "Fourier basis" (for real-valued functions)

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors
    - Dot product with another boolean vector
    - A function in the "Fourier basis" (for real-valued functions)

after changing to ±1 co-ordinates

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors
    - Dot product with another boolean vector
    - A function in the "Fourier basis" (for real-valued functions)
  - Enough to check: is any Fourier coefficient dominant?

after changing to ±1 co-ordinates

Linearity Test

- Is a function table provided close to being linear?
- Test: query f(x), f(y), f(x+y) for random x, y. Check linearity.
- Analysis:
  - Linear boolean function over boolean vectors
    - Dot product with another boolean vector
    - A function in the "Fourier basis" (for real-valued functions)
  - Enough to check: is any Fourier coefficient dominant?
    - Can show that if $\Pr[f(x+y)=f(x)+f(y)] > 1/2 + \epsilon$, then a Fourier coefficient is larger than 2ϵ

after changing to ±1 co-ordinates

New proof

New proof

- Recent development [Dinur'06]

New proof

- Recent development [Dinur'06]
  - A "combinatorial" (as opposed to algebraic) proof of the PCP theorem

New proof

- Recent development [Dinur'06]
  - A "combinatorial" (as opposed to algebraic) proof of the PCP theorem
  - By "gap amplification"

New proof

- Recent development [Dinur'06]
  - A "combinatorial" (as opposed to algebraic) proof of the PCP theorem
  - By "gap amplification"
    - Starting from a small gap (inherent in 3SAT), and amplifying it

New proof

- Recent development [Dinur'06]
  - A "combinatorial" (as opposed to algebraic) proof of the PCP theorem
  - By "gap amplification"
    - Starting from a small gap (inherent in 3SAT), and amplifying it
    - Operations on a constraint graph

New proof

- Recent development [Dinur'06]
  - A "combinatorial" (as opposed to algebraic) proof of the PCP theorem
  - By "gap amplification"
    - Starting from a small gap (inherent in 3SAT), and amplifying it
    - Operations on a constraint graph
    - Uses "expander graphs"

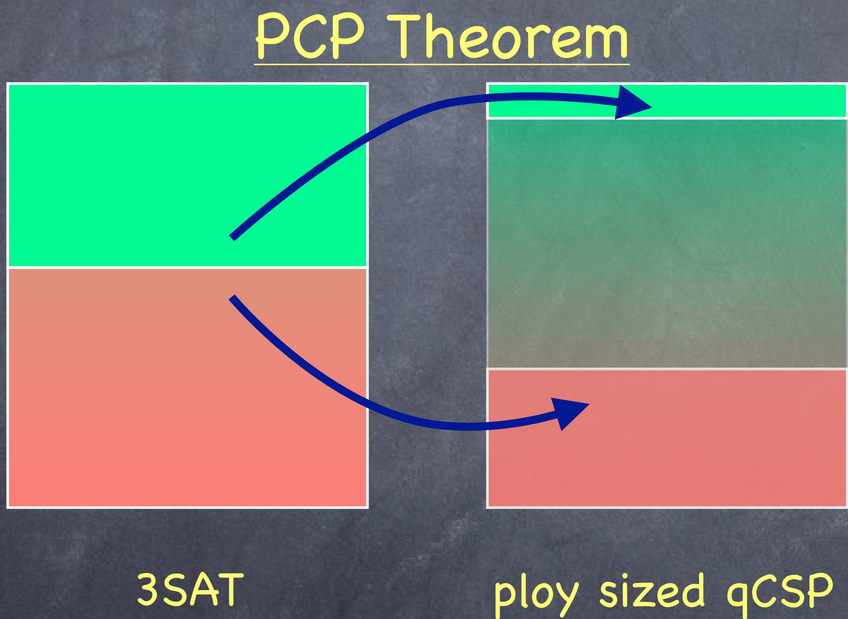
Summary

Summary

- A problem/gap problem has a (log m,q) PCP iff it is efficiently reducible to the gap problem qCSP of size m

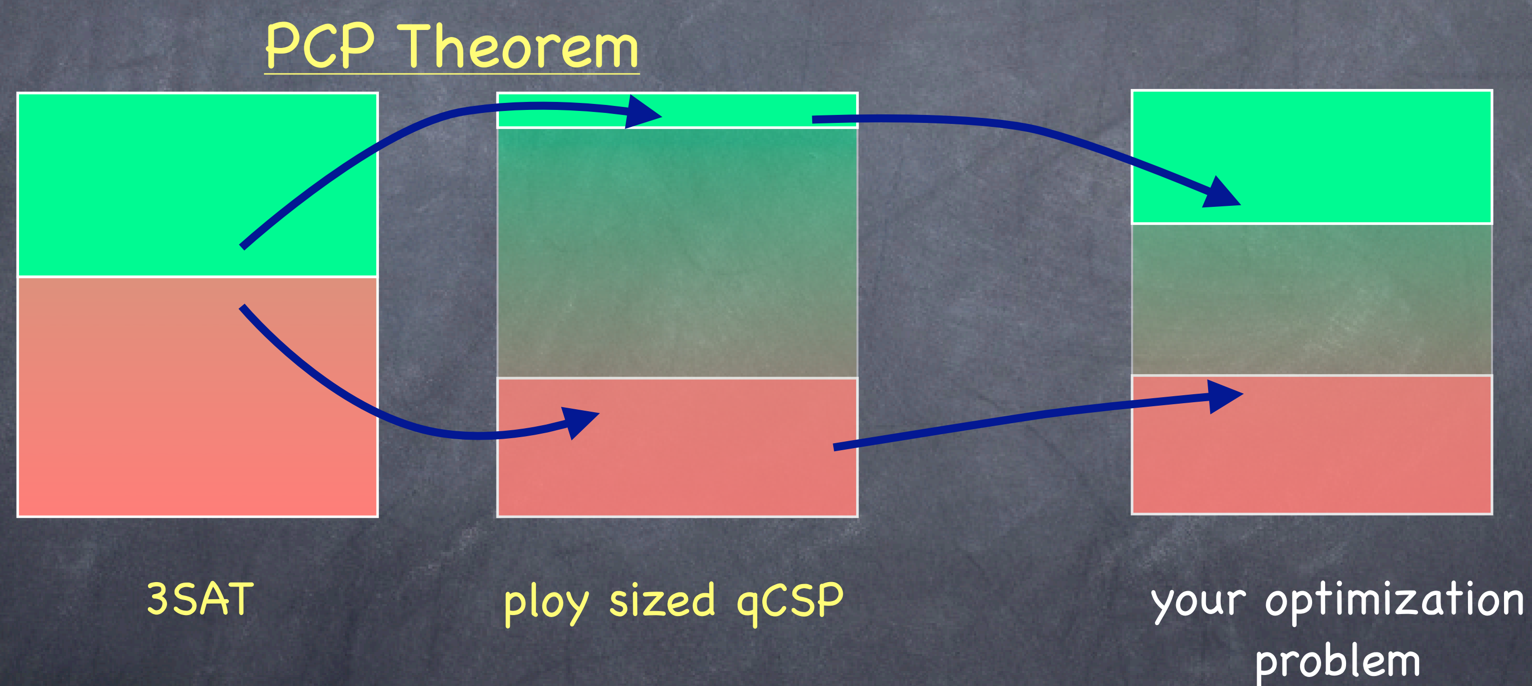
Summary

- A problem/gap problem has a (log m,q) PCP iff it is efficiently reducible to the gap problem qCSP of size m



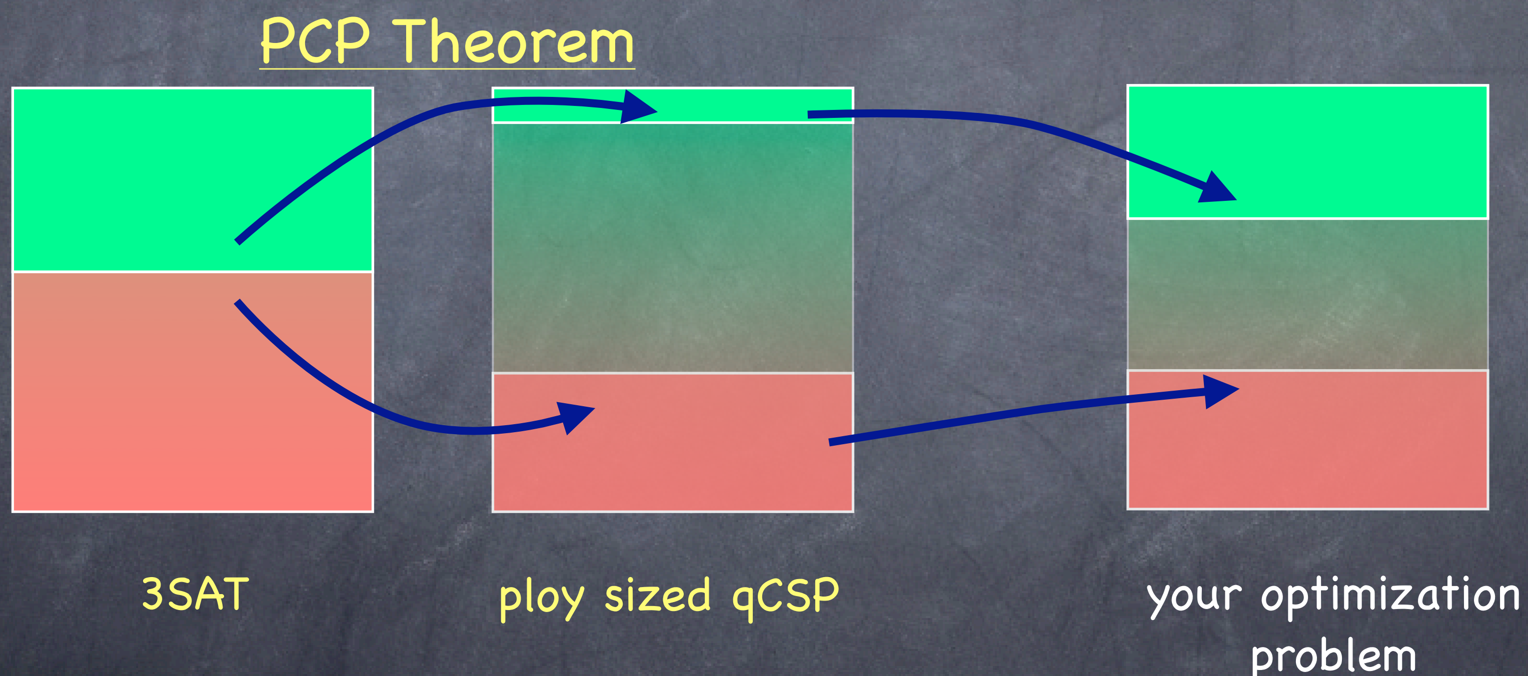
Summary

- A problem/gap problem has a (log m,q) PCP iff it is efficiently reducible to the gap problem qCSP of size m



Summary

- A problem/gap problem has a (log m,q) PCP iff it is efficiently reducible to the gap problem qCSP of size m



- Variants of these reductions to get different hardness results for different approximations