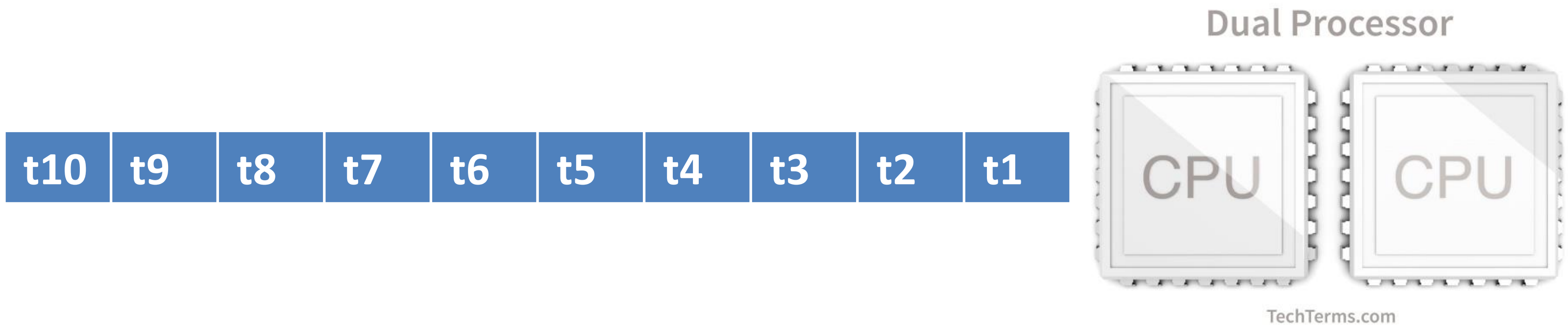
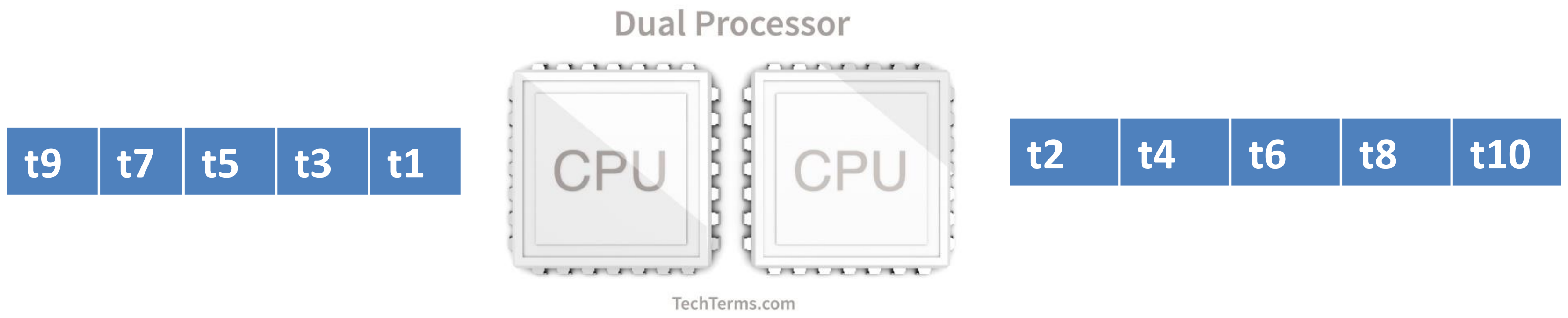
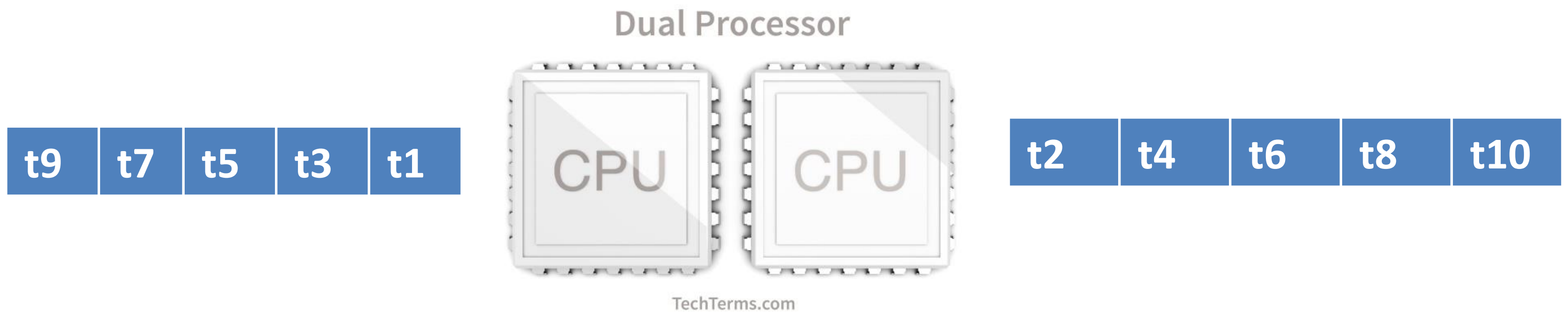
Motivation



- There are a set of tasks and associated size of each task.
- There are two processors of same efficiency
- You have to schedule tasks in such a manner that entire set of tasks finished earliest



- Strategy 1:
  - Even tasks in P1
  - Odd tasks in P2



- Strategy2:
- task will be assigned to the processor which is least loaded 2, 4, 3,3,4,6,8,5,4,15

- Strategy 3:
- First sort the jobs in terms of their size and then apply strategy 2
- 2 2 2 3 3

- Strategy 4:
- Brute force
- Try to partition set of jobs into two sets (s1 and s2)in all possible manner and |size(s1) – size(s2)| is minimum.
- Number of possible partition is 2^N

Reduction

- matching: M will be called a matching of a graph say G if M is a sub-graph of G and no two edges of M share common vertex. And M will be called perfect matching if M is a matching and all vertex of G are part of M.

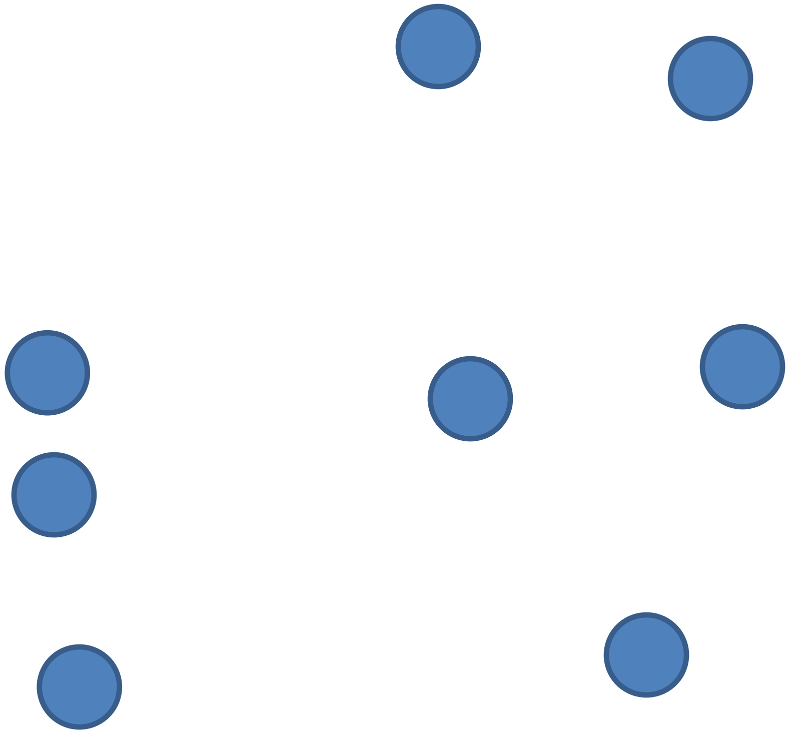
- Problem1:
- Input: Graph G.
- Question: Does G have a perfect matching?

- Problem2:
- Input: Graph G.
- Question: Find a matching of maximum size

- size of a perfect matching is |v|/2 as no vertices are there in matching for more than once and every vertices are there only once so no of edges = no of vertices/2.
- Hence, if we are able to solve second problem then we will be able to solve first problem.

- Will we be able to solve second problem if we are able to solve the first problem.
- Given an algorithm which can comment if there exists any perfect match or not
- Design an algorithm for finding max matching
- Intermediate objective: design an algorithm to find the size of max-matching in G.

- Step1: G_0=G
- i=0
- Step2: Check if G_i does have a perfect matching. If yes then max matching of G_i is |v+i|/2 and max matching of G is |v-i|/2
- If G_i does not have a perfect matching then add a vertex and add edges between new vertex to all vertex of G_i and call this new graph as $G_{i+1.}$
- i=i+1



- Continue to do this until you find a perfect matching with certain value of i.

- Now assume that for i==k we found perfect matching. So our claim is that size of maximum matching of G is |v-k|/2.

- To prove this we have to prove two facts
  - There is a matching of |v-k|/2 of G
  - It is the size of maximum matching

- Consider the G_k graph. It is the graph G along with k other vertices. While each of these k vertices are connected to each other and each of these k vertices is also connected to all other vertices of G.

- So we can think about two partitions, in one partition there are vertices of G (partition1) and in another partition (partition 2) we have newly added k vertices.

- It is also given that G_k has a perfect matching for the lowest k value. Perfect matching involves all vertices exactly once. So there are V+k vertices in this matching out of which k vertices and corresponding edges are newly added. We also claim that this matching only contains edge from partition 1 and edges between partion1 and partition 2 but not from partition2.

- Let us prove by contradiction. Say there is an edge from partition2 in perfect matching. That means if we remove this edge then G_{k-2} is also a perfect matching, however which is a contradiction to our previous assumption that k is the shortest number when we first found that G_k does have a perfect matching.

- So there are edges either from partition1 or from connection of partition 1 and partition2. So if we remove those edges from matching which are connecting between partition1 and partition2 then we are left with |v-k|/2 edges and that is a matching (claim 1 is proved)

- Claim2: let us say there is a matching of size greater than |v-K|/2 or of size |v-k|/2+i. So number of vertices in matching is V-K+2i. Number of vertices which are not included in matching is |V|-|V-k+2i|=k-2i. So if we add k-2i vertices with G that should produce a perfect matching or G_{k-2i} should be a graph with perfect matching. But is a contradiction, as we assume that k is minimum.

- We are now able to design algo with original goal. We know the maximum matching size of G say it is l. Now start with G,
- Step1: For each edge e in G
- Remove e and check if the maximum matching size is still l then continue otherwise add back e and continue

Hamiltonian Cycle and Hamiltonian Path

- **Definition**:
  - **Hamiltonian Path (HP) of a Graph**: It is a path of graph which spans over all n vertices of graph exactly once
  - **Hamiltonian Cycle (HC)** : It is a cycle which covers all the vertices.

- **Problems:**
  - **Problem1**: Given a graph G find out if there exists any Hamiltonian cycle (HC) or not.
  - **Problem 2**: Given a graph G find out if there exists any Hamiltonian path or not.
  - **Problem 3**: If we have an efficient algorithm of problem1(HC) then we can design an efficient algorithm for problem2 (HP).
  - **Problem 4**: We are given an efficient algorithm for finding Hamiltonian path. Can we use this algorithm to find out if there is any Hamiltonian cycle in Graph.

Problem 3

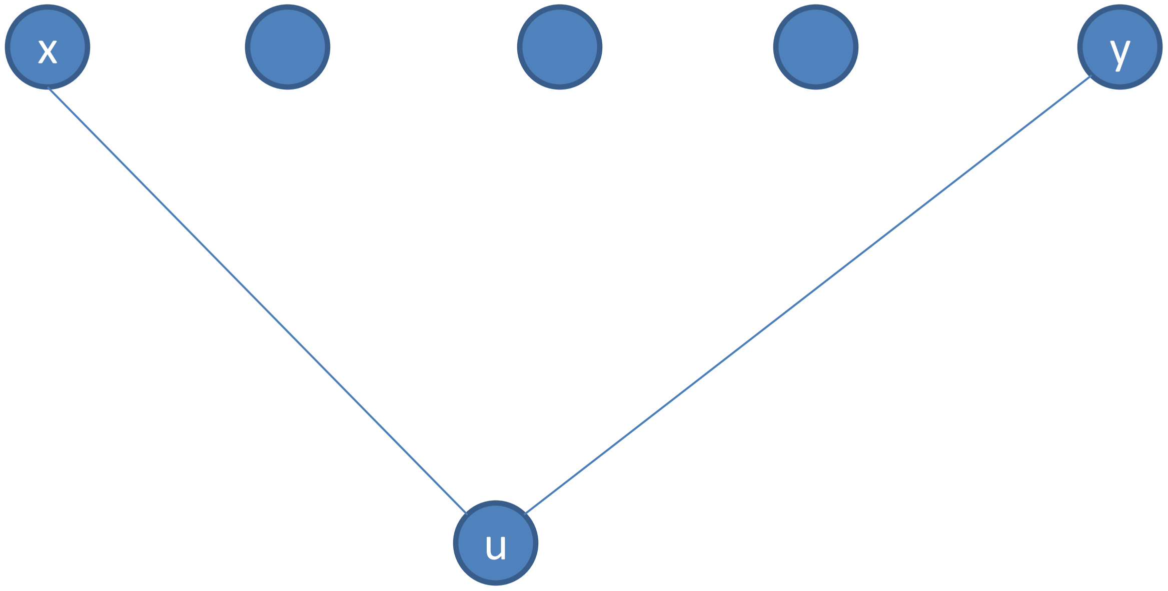
- Let us assume that given graph is G and the efficient HC algorithm is available. Now, we have to comment if there is any HP or not.

- In HC, we use graph G as input and answer can be
  - Yes
  - No

- If the answer is yes, that means there is a Hamiltonian cycle in G. By definition of Hamiltonian path, G is also having Hamiltonian Path.

- In the other case when G does not have a Hamiltonian cycle, it might contain a Hamiltonian Path or may not have any Hamiltonian Path. So how to conclude if G is having any Hamiltonian path or not when there is no Hamiltonian cycle.

- We transform G into G’ by adding an additional vertex say U and adding edges from U to all the vertices in G. Now we use graph G’ as input for HC algorithm. There are two possibilities
  - G’ is having a Hamiltonian cycle
  - G’ does not have any Hamiltonian cycle


x
y
u

- Consider case i): We claim that G is having a Hamiltonian path. Consider the Hamiltonian cycle. This cycle also contains vertex U and it is connected to two other vertices say X and Y. If we remove UX and UY edges and vertex U then there is a path from vertex X to vertex Y which passes through all vertex of graph G exactly once and all those edges are there in G. So there is a Hamiltonian path in G.

- Case ii) We claim that G does not have any Hamiltonian path. Let us prove by contradiction. Say there is a Hamiltonian path from where starting vertex is X and end vertex is Y. When we add U in G’, there is edge between X and U as well as Y and U. So G’ should have a Hamiltonian cycle which is a contradiction. So G does not have any Hamiltonian path.
-

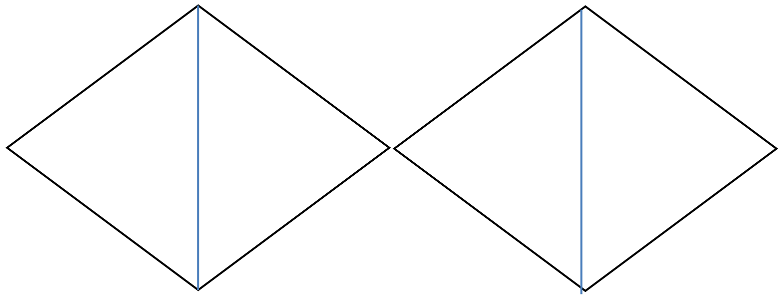
Problem 4

- We are given an efficient algorithm for finding Hamiltonian path. Can we use this algorithm to find out if there is any Hamiltonian cycle in Graph.
- Let us say input graph is G and algorithm for finding Hamiltonian Path is HP. We have to use this HP for efficient HC design.
- We use the graph G as input in HP, there are two possibilities

- Case i) There is a Hamiltonian Path in G
- Case ii) There is no Hamiltonian path in G
- For Case ii) From the definition of Hamiltonian path and cycle we can comment that there is no Hamiltonian cycle.
- Case i) When there is a Hamiltonian path, either graph G contains a Hamiltonian cycle or it might not have any Hamiltonian cycle.

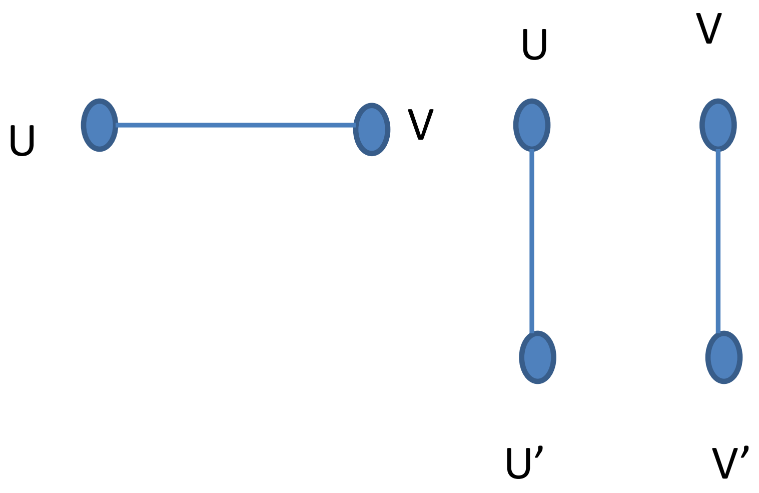
- Let us say that we will consider all the edges of graph G one by one and check if there is Hamiltonian Path after removal of that edge.
- If there is a Hamiltonian cycle in graph G, then removal of any single edge could not result in no Hamiltonian path.
- So if after removal of each edge we found that there exists Hamiltonian path then we can say that G contains a Hamiltonian cycle.

Counter Example



There is Hamiltonian path in above graph, even there will be a Hamiltonian Path even after removal of any one edge from the graph. However, there is no Hamiltonian cycle. As removal of bridge vertex will leave the graph disconnected which would not be possible for a graph having Hamiltonian cycle.

- Where was the flaw?
- If there is Hamiltonian cycle then removal of any single edge would result in another graph which contains Hamiltonian path but the opposite implication is not true, that if after removal of any single edge graph contains a Hamiltonian path it does not imply that there was Hamiltonian cycle. If there are alternative Hamiltonian paths in Graph, it would happen.



Modify G to G' in this way. Consider any edge say UV of graph G. Add two more vertices U' and V' add edges between U and U' as well as V and V' and remove edge UV. Now check if G' is having any Hamiltonian Path. If for any UV if we find that G' is having a Hamiltonian path then we claim that G is having a Hamiltonian cycle.

Let us consider that G' is having a Hamiltonian path. As U' and V' nodes are nodes with degree 1, Hamiltonian Path will look like U'U……VV'. So there is a Path between U to V containing all vertices of G. There is also an edge between U and V in G so there is a Hamiltonian cycle in G.

Types of Problem

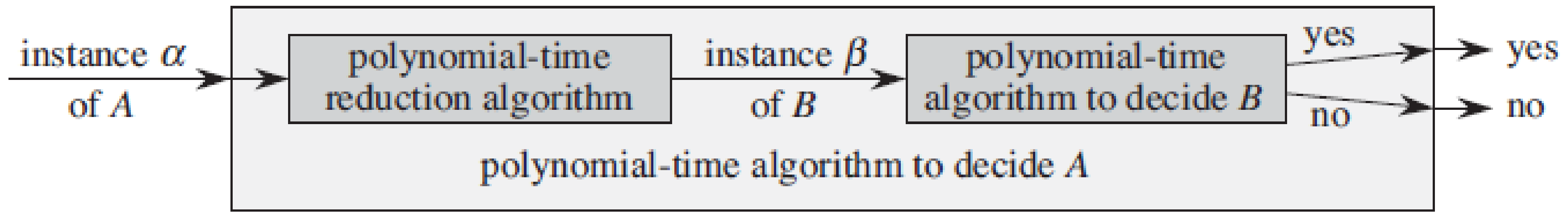
- There are two types of problem
  - Which outputs in terms of yes/no – need a single bit: decision problem
  - Which outputs by using many bits, for example by finding maximum matching: search problem

- For every search problem there is a corresponding decision problem which are closely related. If one is easy to solve then other will be also easy to solve and if one is hard to solve then other one is also hard to solve.

Prover	Verifier
All powerful	Limited resource, does not trust prover
For example if prover say there is Hamiltonian cycle in the graph	Verifier will not trust
If prover provides the cycle	Verifier can verify it easily

A decision problem will be said in NP if for all Yes Inputs, there is a proof/advise/certificate which can be verified in polynomial time that input is indeed a Yes input.

Reduction


instance α
of A
polynomial-time
reduction algorithm
instance β
of B
polynomial-time
algorithm to decide B
yes
no
yes
no
polynomial-time algorithm to decide A

Circuit Satisfiability

- Boolean Variable: value is either Yes/No
- If X is a Boolean variable then ¬X is its negation.
- A literal is either a Boolean variable in its true form or its negation
- Clause is or of literals. If any literal is true then clause will be evaluated as true.
- C=l1 V l2 V l3 V l4

- A Boolean formulae in CNF (Conjunctive Normal Form) is AND of clauses
- C1 AND C2 AND C3
- When all clauses are true then formulae will be evaluated as true, otherwise it will be evaluated as false.
- A Boolean formula will be said satisfiable if we find assignments of all variable which evaluates to be true.

- Input: Boolean formulae f is in CNF
- Question: Is f satisfiable?
- SAT is an NP problem.
- Cook Theorem: If SAT has an efficient solution then so does every other problem in NP.
- $X_0 \leq_p$ Circit-SAT for every $X_0 \in$ NP
- Circit-SAT $\leq_p$ Y then what can we comment about Y?

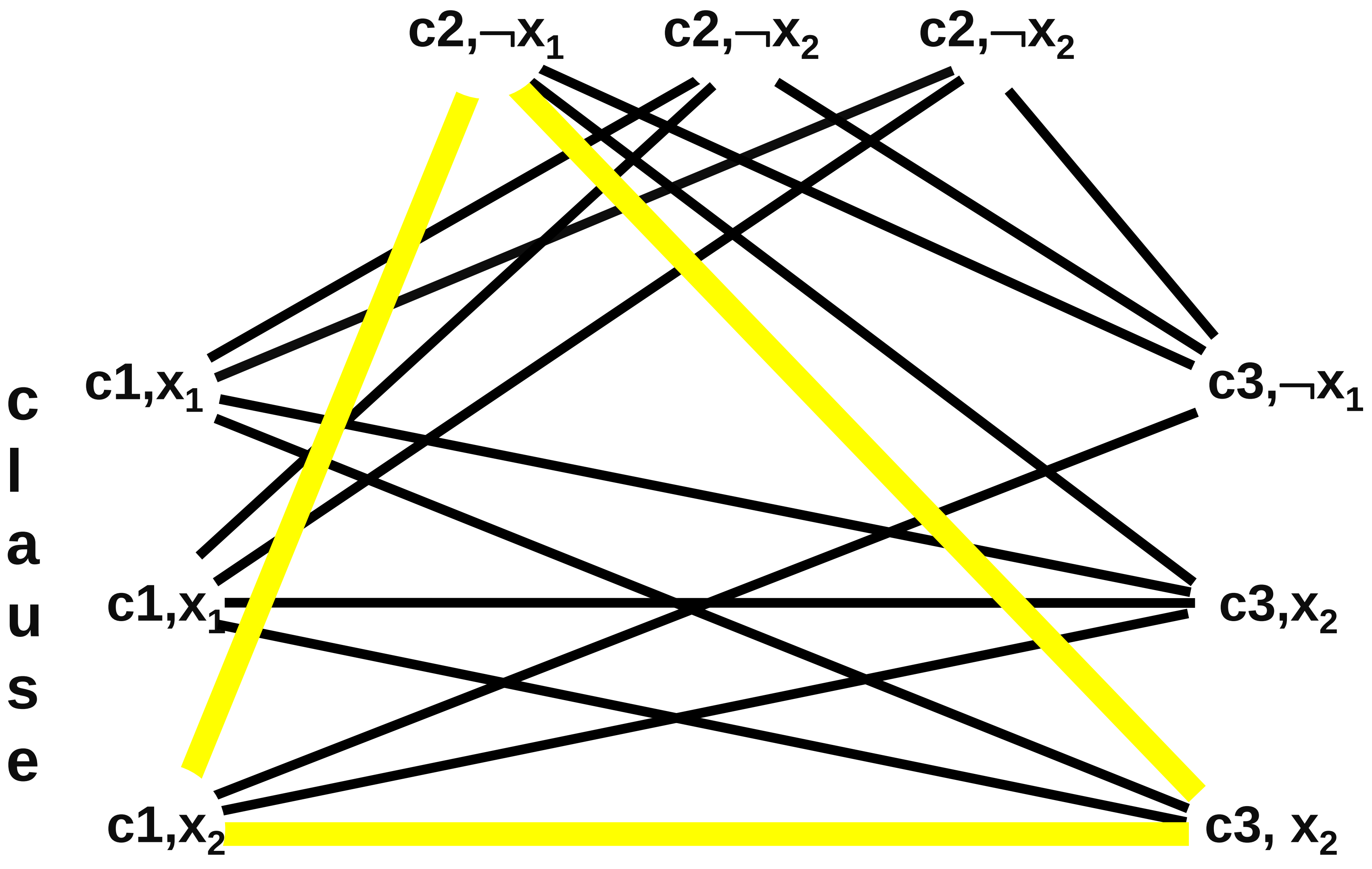
- A clique in a graph G=(V,E), is a subset U of V such that for every two vertices u1, u2 belongs to U there is an edge between u1 and u2.

- Input: Graph G and an integer k
- Question: Does G has a clique of size k?

- There will be a vertex for every clause-literal pair
- Two vertex (c1,l1), (c2,l2) will be connected by an edge if c1!=c2 and l2!= ¬l1

c1 $\wedge$c2 $\wedge$c3

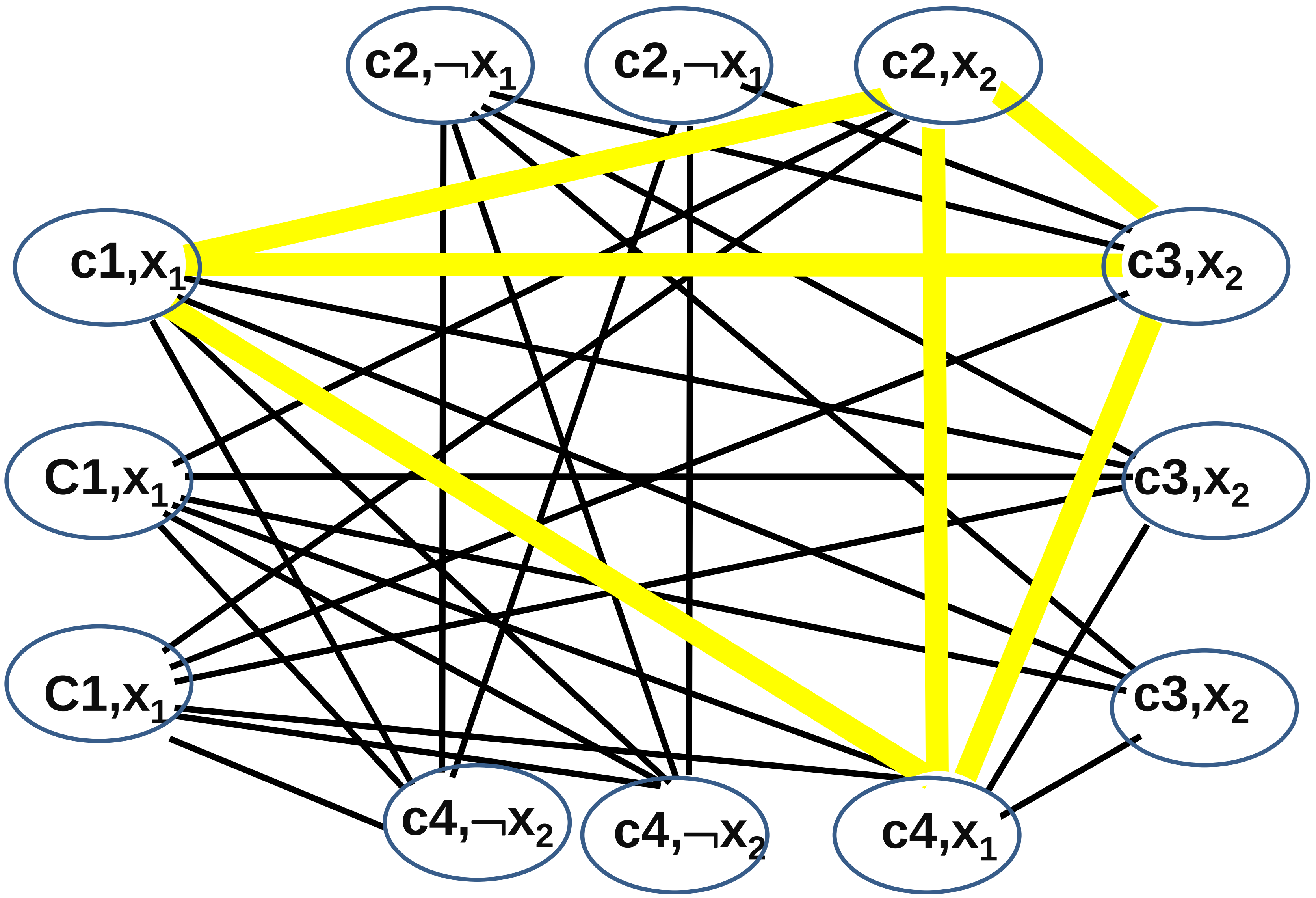
$= (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$



nodes = 3(# clauses)

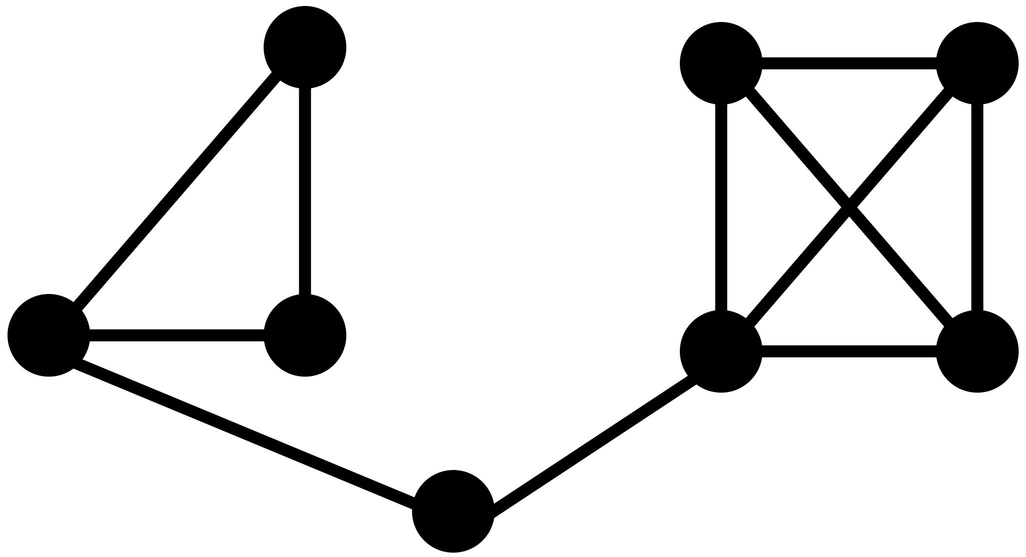
k = #clauses

$$c1 \wedge c2 \wedge c3 \wedge c4 = (x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee x_2) \wedge (x_2 \vee x_2 \vee x_2) \wedge (\neg x_2 \vee \neg x_2 \vee x_1)$$



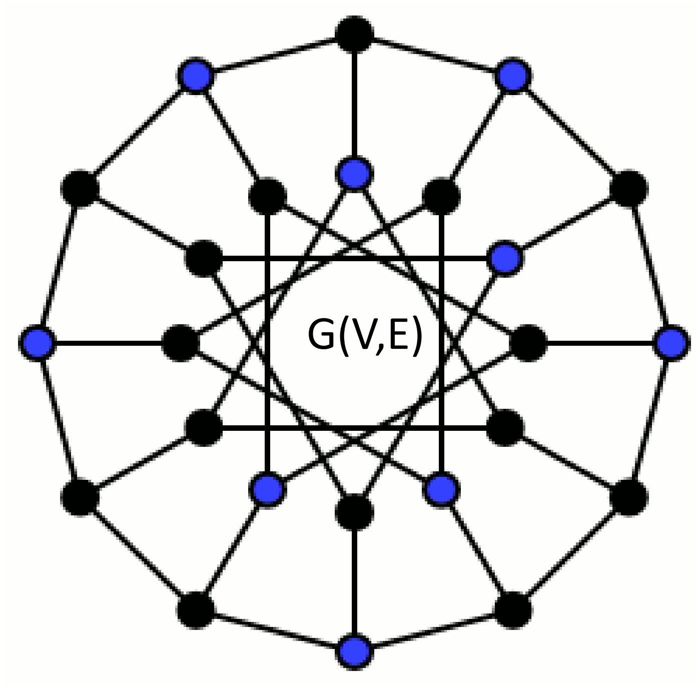
Independent Set

- An independent set is a set of nodes with no edges between them



This graph contains an independent set of size 3

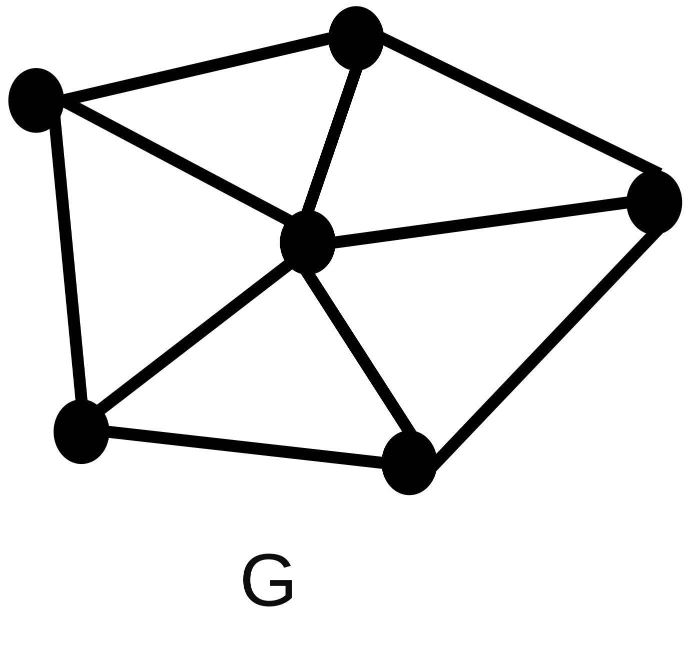
Independent Set

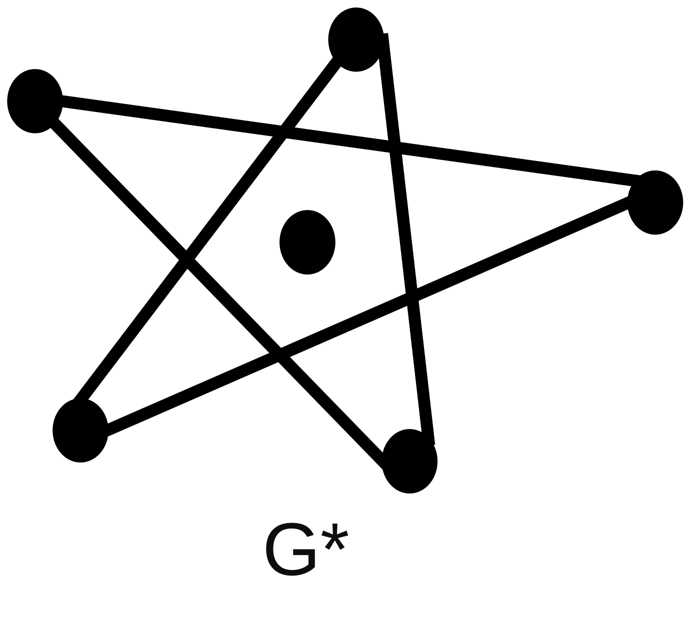


Problem: Given a graph G and k, is there a size k independent set?

Complement of G

Given a graph G, let G*, the complement of G, be the graph such that
two nodes are connected in G* if and only if the corresponding nodes are not connected in G



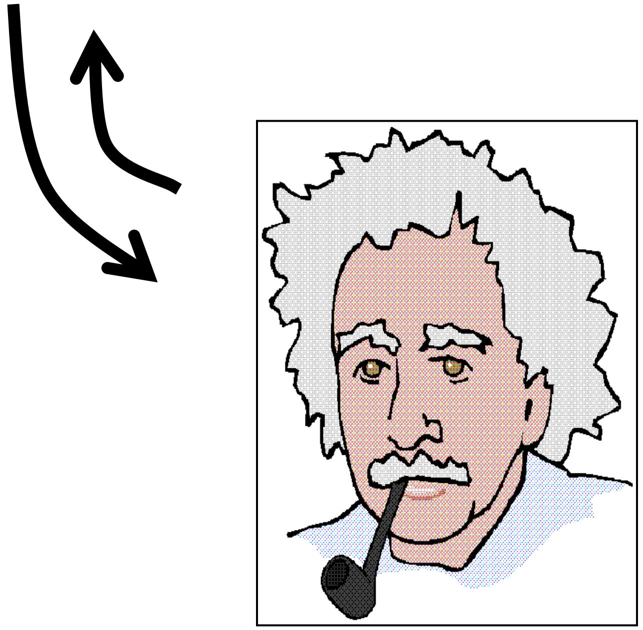


Key Observation

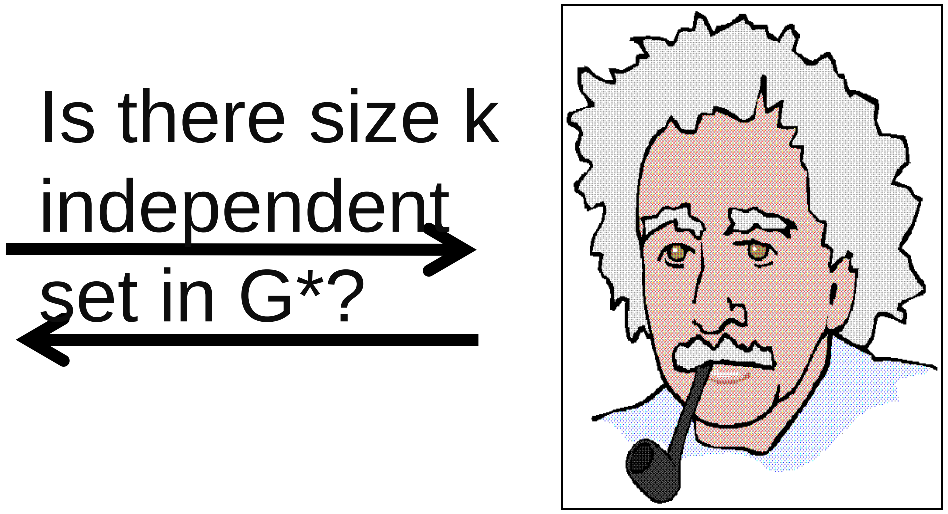
- For a graph G, vertex set S is an independent set if and only if S is clique in G*.

Let G be an n-node graph

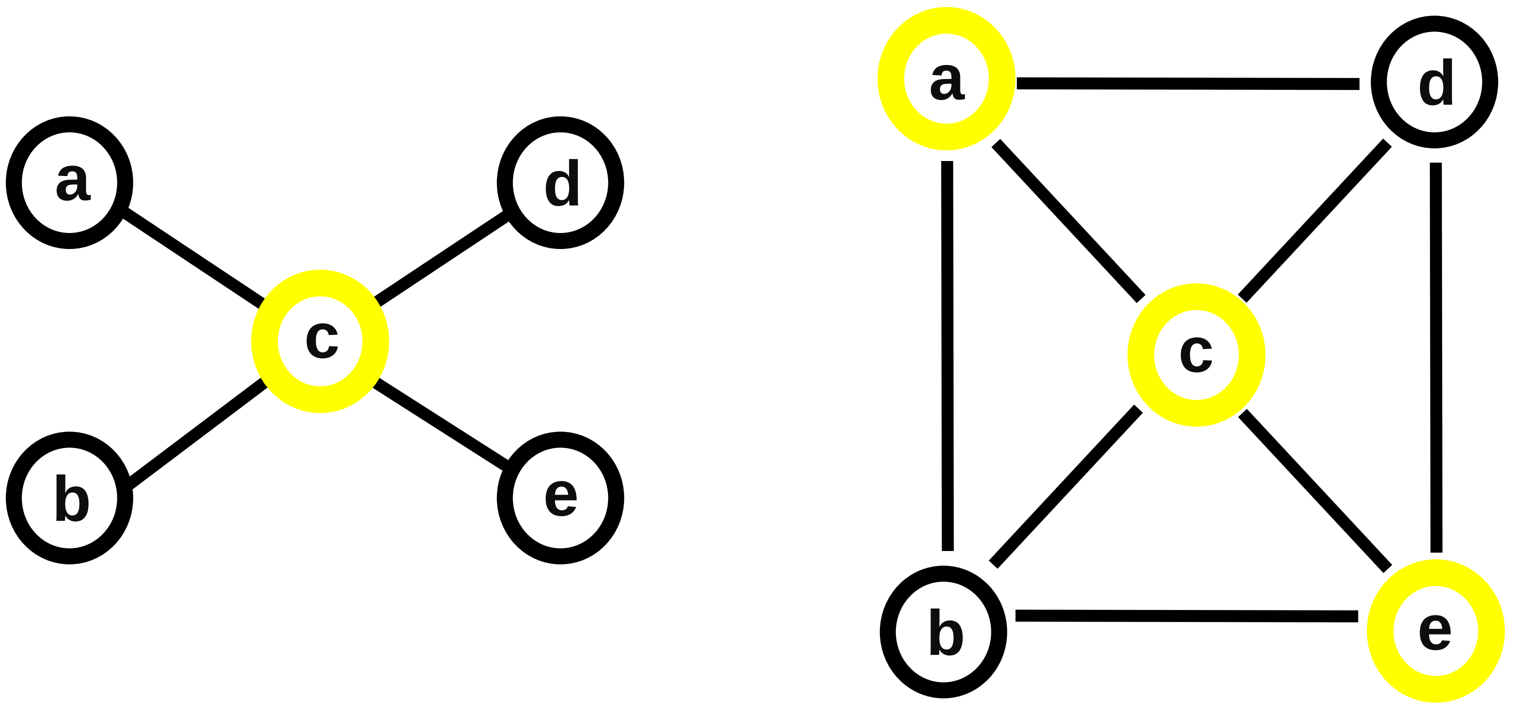
Is there a size k clique in G?



Is there size k independent set in G*?



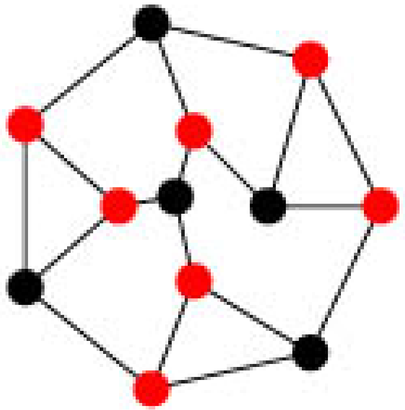
VERTEX COVER



vertex cover = set of nodes that cover all edges

Vertex Cover is NP Complete

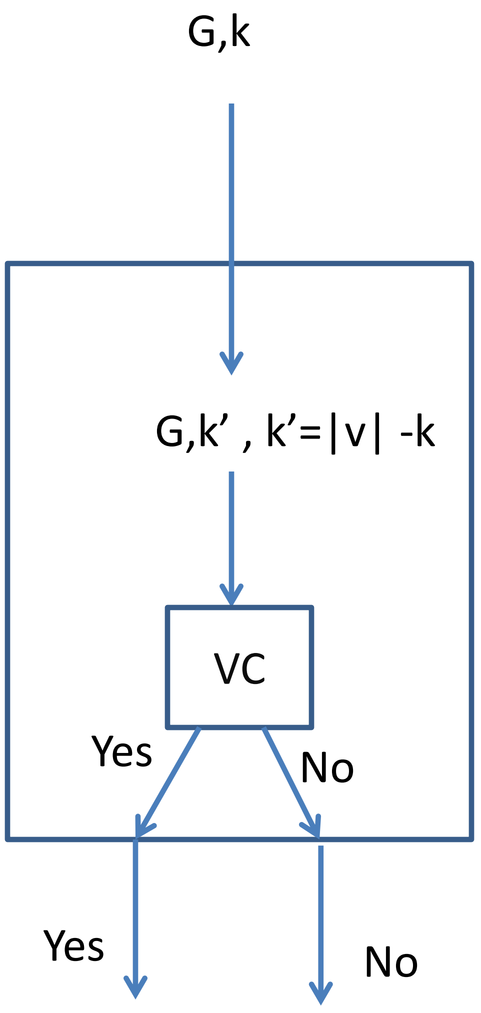
- Given a Graph G(V,E), decide if there is k vertex such that every edge is covered by one of them?

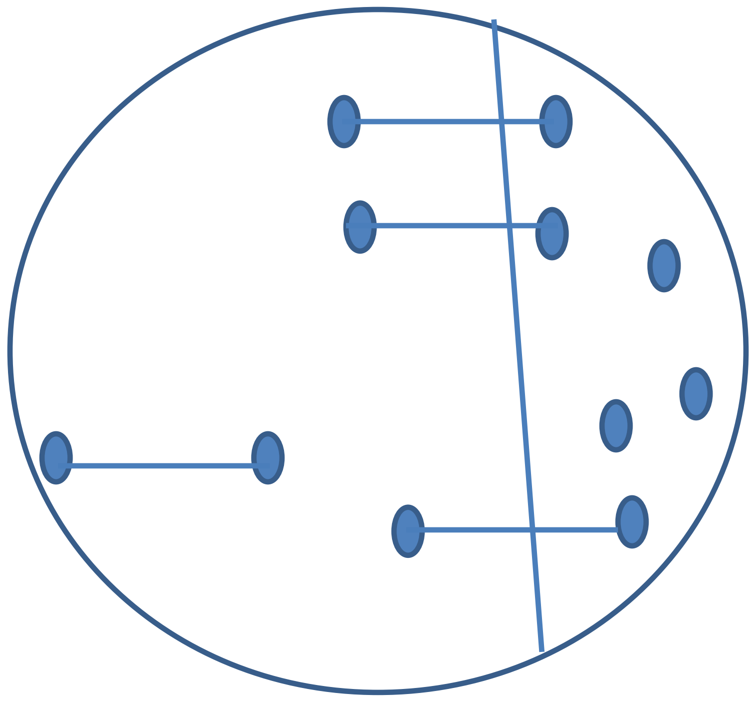


K-Indep Set $\leq_P$ Vertex Cover

- Assume we have a polynomial time algorithm for Vertex cover
- Can you develop a polynomial time algorithm for independent set
- Given a graph G and an integer k => does graph G has an independent set of size k

K-Indep Set $\leq_P$ Vertex Cover



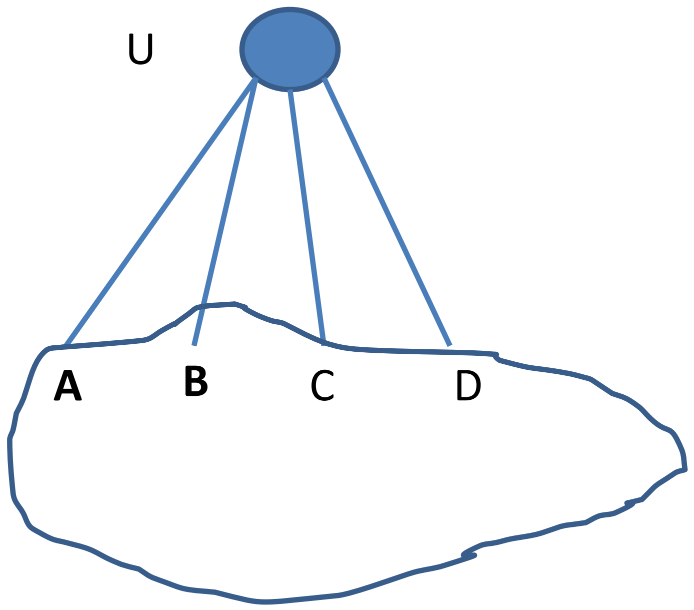


$\leq$ 3VC

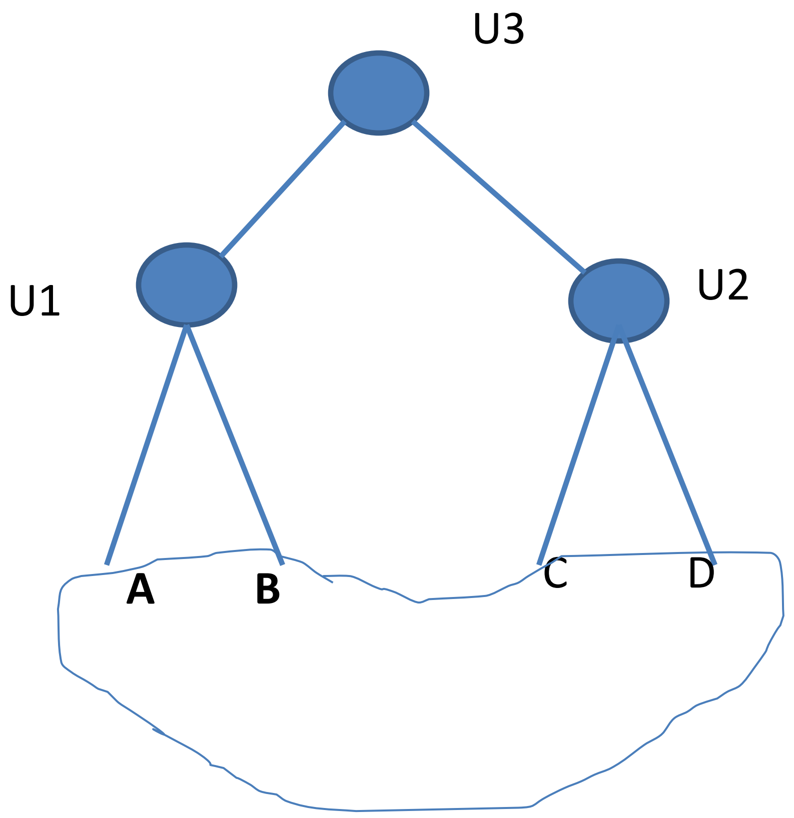
- Input: A graph G with maximum degree 3, and positive integer k
- Question: If graph G does have a vertex cover of size k?

Vertex Cover $\leq_P$ 3VC

Degree Reduction



Graph G



Graph G'

- Effect of **a** split
  - Graph G has a vertex cover of size k iff graph G’ has a vertex cover of size k+1
- => case i) let us say u is in vertex cover of G then remove u and put u1 and u2 that will be a vertex cover of G’
- Case ii) u is not in vertex cover of G then put u3 and that will be vertex cover for G’

- <= case i) u3 in vertex cover of G’ but not u1 and u2 then remove u3 that will become a vertex cover of G
- Case ii) at least two nodes of (u1, u2 and u3) are in vertex cover of G’ then remove both the nodes and add u that will be vertex cover of G

Exact Cover (XC)

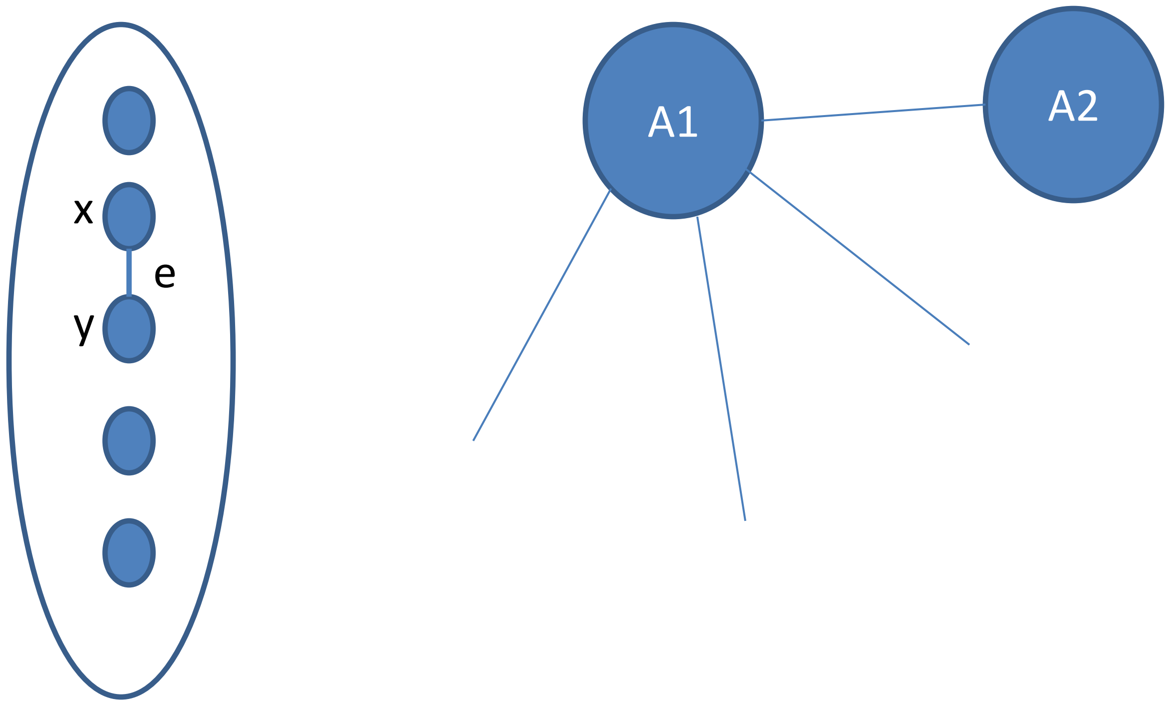
- Input: Collection of subset of S : A1, A2,…Am and an integer k
- Question: Is there any specific sub-collection of size k which are non-overlapping and union of those is equal to S?
- a_i, a_{i+1}, …,a_{i+k-1}

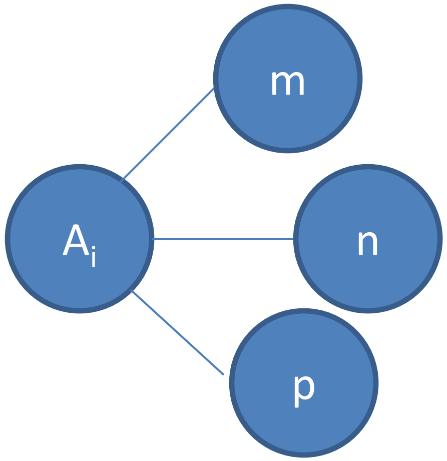
$$VC \leq_{P} XC$$

- Vertices to cover all edges
- Subset to cover all elements

- Vertices $\Longleftrightarrow$ Subsets
- Edges $\Longleftrightarrow$ elements
- S= E(G)
- A_i = (set of all edges incident on vertex A_i)

- $S = E(G) \cup \{x_1, x_2, \ldots, x_k\}$
- $A_1 \cup \{x_1\}$, $A_2 \cup \{x_1\}$, ….,$A_n \cup \{x_1\}$
- $A_1 \cup \{x_2\}$, $A_2 \cup \{x_2\}$, …., $A_n \cup \{x_2\}$
- ……………………………………………
- $A_1 \cup \{x_k\}$, $A_2 \cup \{x_k\}$, …., $A_n \cup \{x_k\}$


x
e
y
A1
A2



- A_iU{{m,n,p},{m,n},{m,p},{n,p},{m},{n},{p}}U{X_1, …X_k)

Subset-Sum

- Input: There is a set S with m elements. Elements are of size $s_1, s_2, \ldots s_m$. You are also given an integer B.
- Problem: Can we find out a subset T of S such that total size of T is B.

$$\sum_{e \in T} size(e) = B$$

- Given a polynomial time solution for subset sum, can we build a polynomial time algorithm for exact cover?
- XC: A_1, A_2, …A_m are there we need to choose such a subset T so that all elements of S are covered exactly once. We are assuming S is universe and distinct elements are x_1,x_2,..x_k.
- Subset-Sum(SS): S is a set with m elements with sizes s_1,…s_m. Can we choose a subset of S so that total size of that subset is B.
- Subset of XC $\leftrightarrow$ element of SS.
- Exact Cover $\leftrightarrow$ Subset of size B

- Multiset M: Multiset is a set where repetition of element is allowed. Also assume that an element can be repeated for at most l times.
- Can we assign some size to each unique elements of M such that if we need to choose a subset from M of a specific size then all unique elements must be chosen exactly once?

- Let’s say we have only two unique elements x1 and x2 (each elements may have at most l copies)
- Now assume that size of x1=1 then what should be size of x2 to fulfill the mentioned criteria?
- Can it be 2/3?
- It can be l
- So if we want to choose a subset of size (1+l) then we must choose one copy of x1 and one copy of x2 provided that l+1<2l

- Can we generalize it?
- X1 X2 ….Xk
- 1 l l^{k-1}

$$\begin{tikzcd} X1 \arrow[d] & X2 \arrow[d] & \text{….}Xk \arrow[d] \\ 1 & l & l^{k-1} \end{tikzcd}$$

- $B=1+l+\ldots+l^{k-1}$
- $A_i \Rightarrow s_i$ (add size of all elements belongs to A_i)

Partition

- T: is a set with elements of size t1, t2, t3,..,tn
- Question: Can we partition this set into two equal size subset?
- To prove this problem is NPC we have to prove two things
- i) Problem is NP
- Ii) Problem is NP-Hard

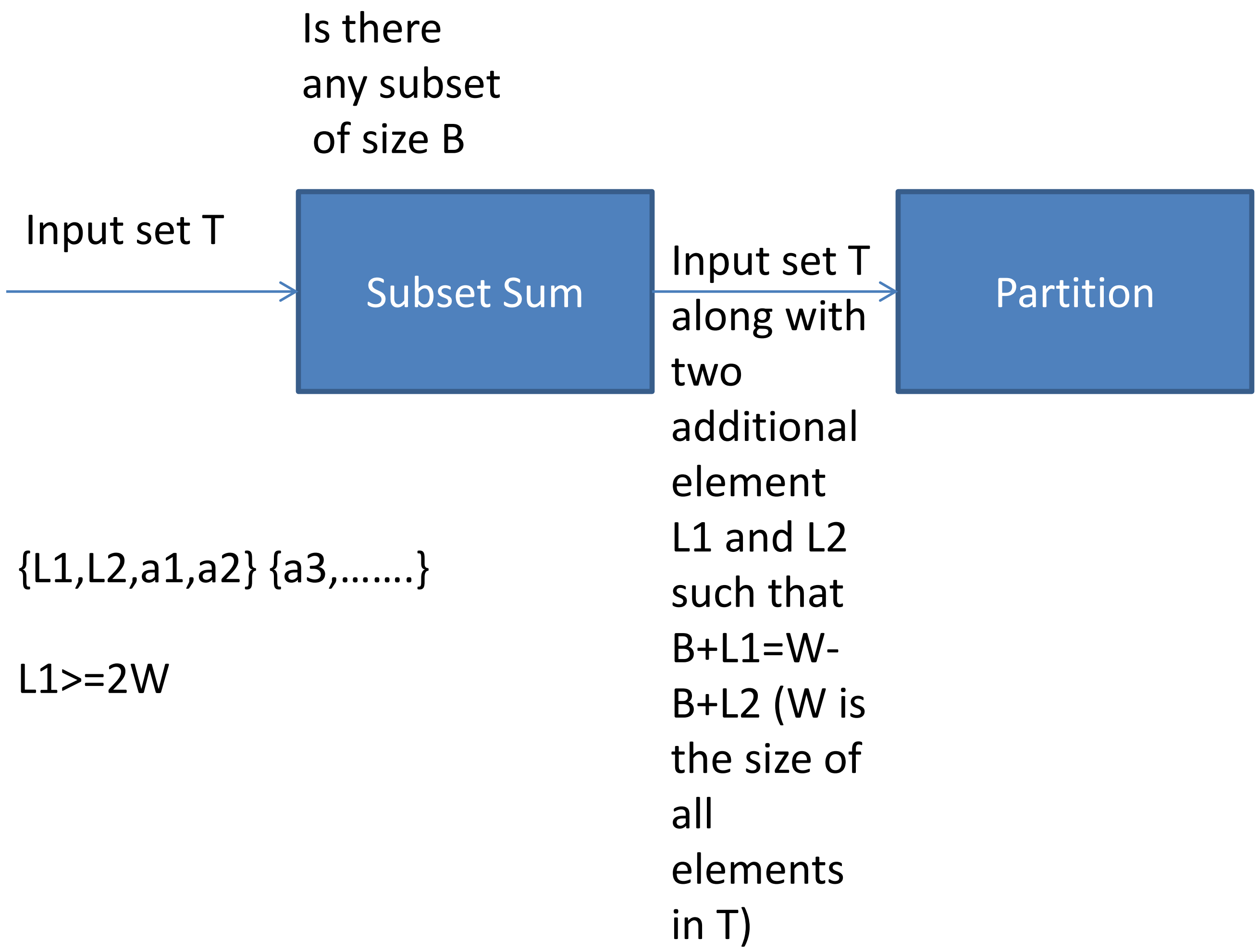
- Problem is NP: A problem will be in NP if it has following two features

i) It is a decision problem
ii) Given a proof/certificate can be verified in polynomial time.

It’s a decision problem as we are looking for a yes/no answer.

- Certificate can be in the form of two subset which we can check if they are of equal size in polynomial time.

- Hence this problem belongs to NP


Is there any subset of size B
Input set T
Subset Sum
Input set T along with two additional element L1 and L2 such that B+L1=W-B+L2 (W is the size of all elements in T)
Partition

{L1,L2,a1,a2} {a3,…….}

L1>=2W