KombSøg Eksamensnoter 2017

Anders H Pedersen

July 10, 2017

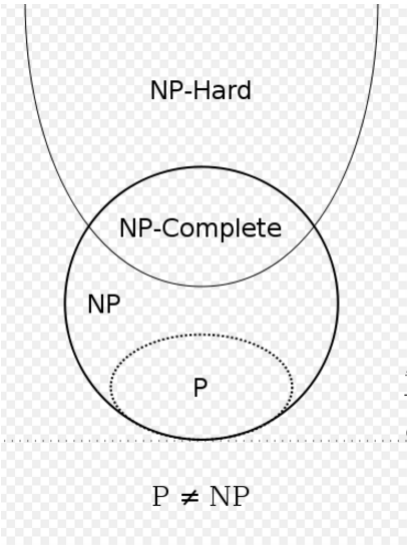
Contents

1 P, NP and NPC.

1.1 Beslutningsproblemer og stand-in sprog

I dette kursus har vi fokuseret på beslutningsproblemer, dvs. problemer hvor outputtet er yes/no.
Og vi har indført såkaldte stand-in sprog for de oprindelige sprog. Eksempelvis max cut, som er et optimerings problem hvor vi leder efte wr det største cut. I stand-in problemet får vi udover grafen også givet et target k, og vi spørger om der findes et cut af størrelse mindst k. Ideen er så at vi kan vise at stand-in sproget ikke har en effektiv algoritme, og dermed har det originale problem heller ikke stand-in sproget ikke er sværere end det originale(Hvis nogen påstår at det er nemmere så kan vi bruge algoritmen for det og sammenligne med vores target k, og dermed få et rigtigt svar).

1.2 P, NP, NPC



P er klassen af problemer som kan løses i poly-tid af en TM. Og når vi skal argumentere for at et problem er i P, så beskriver vi en effektiv algoritmer og referere til **Polynomial Church-Turing thesis** som siger noget om at noget kan løses af en TM i poly-tid hvis og kun hvis det kan i poly-tid af en "fornuftig beregningsmodel".

NP er klassen af problemer hvor løsninger kan verificeres i poly-tid. P er en delmængde af NP: hvis vi kan finde en løsning i polynomiel tid så kan en given løsning tjekkes ved blot at løse problemet i poly-tid og ignorere et eventuelt certifikat. Når vi skal argumentere for at et problem er i NP skal vi således blot beskrive en poly-tids algoritme som verificerer løsninger.

NPC er klassen af problemer som er i NP og som samtidig er NP-hard. At et problem er NP-Hard vil sige, at alle problemer i NP kan reduceres til dette. NPC er en særlig interessant klasse da den indeholder mange problemer som vi gerne vil løse, men også fordi at vi kan bruge den til at vise at nogle problemer bare ikke har en effektiv algoritme (under antagelse at $P \neq NP$). Dette kan vi gøre med en reduktion fra et problem som vi allerede ved at i NPC.

1.3 Reduktioner

Vi har talt meget om reduktioner i kurset. Reduktioner bruges i dette kursus til at vise nogle ting om et sprog.
En reduktion r mapper instanser fra et sprog til et andet, og vi skriver $L_1 \leqslant L_2$ hvis vi kan beskrive en valid reduktion. Den er gyldig hvis udførselstiden er polynomiel, og hvis der gælder at $\forall x : x \in L_1 \iff r(x) \in L_2$. Vi bruger dette til at vise at nogle problemer er NP-complete. Ideen er, at hvis vi ved at der ikke findes en effektiv algoritme som løser L_1 (under antagelse at $P \neq NP$), og vi samtidig kan reducere L_1 til L_2, så kan der heller ikke findes en effektiv algoritme til at løse L_2 (da vi ellers lige har beskrevet en poly tids algoritme til L_1.)

Kører i polytid da vi for hver iteration mindsker en clause med 1 literal indtil vi er færdige.

$\Longrightarrow$: Givet en satisfying assignment for F skal vi vise at F' også er satisfied. Vi ser på hvilke dele af F som gøre den satisfied, hvis det er l_1 eller l_2 så sætter vi y = 0 i F'. Hvis l_1 og l_2 ikke er variablerne som tilfredsstiller f, så må det være en af literals i A, så sætter vi y = 1.

$\Longleftarrow$: Givet at F' er tilfredsstillet så er F selvfølgelig også uanset sandhedsværdien af y.

1.4 3SAT $\leqslant$ HAMILTONIAN PATH

Theorem 9.7: HAMILTONIAN PATH is NP-Complete

We prove this by reducing from 3SAT, which we know is NP-complete.

We construct a graph G such that G has a Hamiltonian path if and only if the input 3sat formula has a satisfying assignment.

For each variable we add a choice gadget and join them. For each clause we add a triangle where each edge corresponds to a literal in the clause. For each edge in the triangle, we add a consistency gadget from the edge, to the edge in the corresponding choice gadget. All nodes in all triangles belong to a clique together with the edge following the last choice gadget, and some other node which is the only one adjacent to the last edge in our path (see figure 6.)

The important thing is that all triangle have at least one edge visited (or actually the 4 nodes on the edges) by the time we get to the clique member after the last choice gadget. If this is not the case then we cannot visit all nodes in a triangle without visiting one of the clique members twice. Recall that edges in the triangles represent literals in the clause - the edges we visit via the choice gadgets are the one for which the corresponding literal make the clause true, and so if no edge is visited, then the clause is false. This is consistent with the fact that we cannot make a hamiltonian cycle.

Figure 5 show exactly what happens with the consistency gadget. Our path goes through nodes 1,2,3,4... and leaves at 12. It is clear that none of the black nodes are visited.

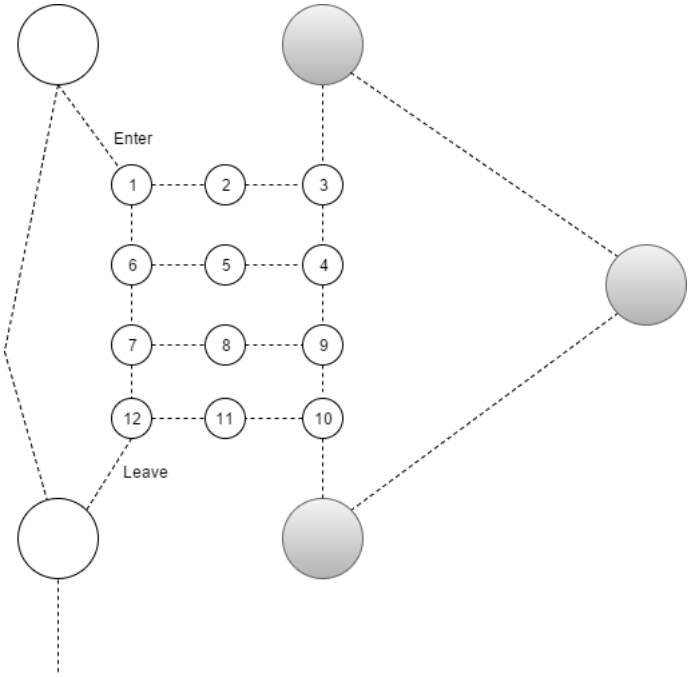


Figure 5: Zooming in on the consistency gadget between the choice gadget and the triangle.

Example:
Given a 3SAT formula $f = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
We can construct a graph similar to the one in figure 6. For the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 0$, we can obtain a hamiltonian path by starting at 1, and follwing the pink line to 2.

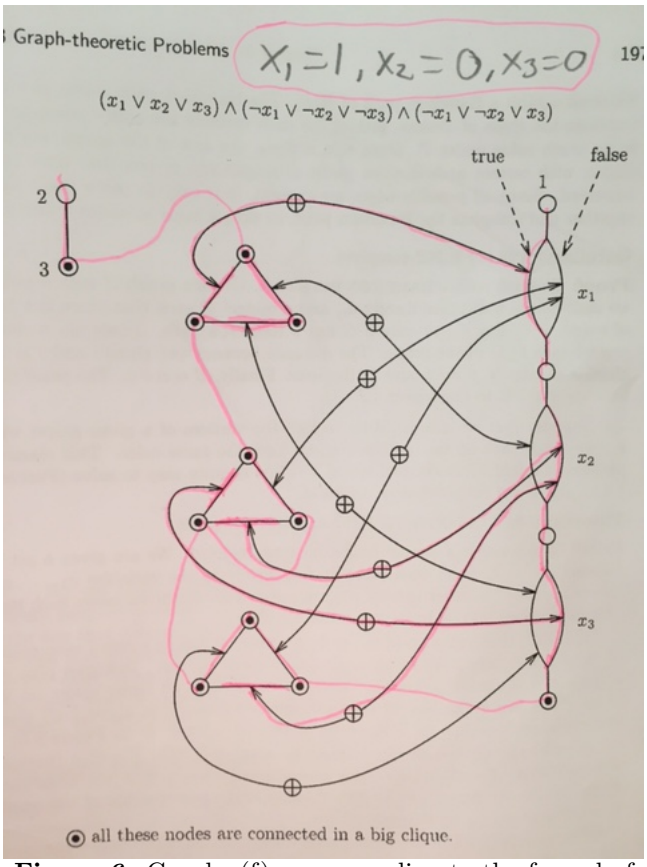


Figure 6: Graph r(f) corresponding to the formula f

2 Cook's theorem and the complexity of variants of SAT.

2.1 Beslutningsproblemer og stand-in sprog

2.2 P, NP, NPC

2.3 Varianter af SAT

1. $SAT \in NPC$
2. $2SAT \in P$
3. $3SAT \in NPC$
4. $NAESAT \in NPC$
5. $MAX2SAT \in NPC$

2.4 Cook: $L \in NP \leqslant CircuitSAT$

2.5 Circuit SAT $\leqslant$ SAT

2.6 Circuit SAT $\leqslant$ NAESAT

Genbrug den forrige reduktion og tilføj literals til clauses med 1 og 2 literals (TODO SE BOG)

2.7 SAT $\leqslant$ 3SAT

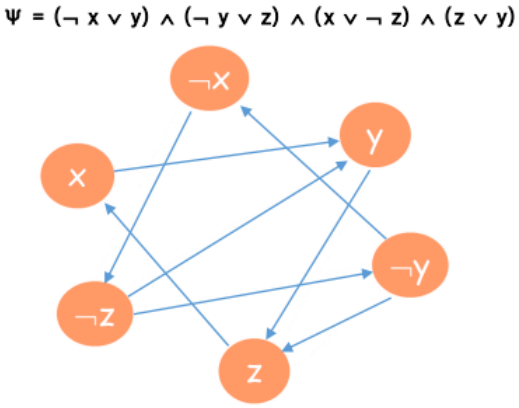
Givet en CNF formel som har clauses med mere eller mindre end 3 literals skal være gøre følgende for hver clause:

1. Hvis en clause har 1 eller 2 literals, så kopierer vi bare en literal 2 eller 1 gang hhv. Eksempelvis $(x \vee y)$ bliver til $(x \vee y \vee y)$ dette ændrer ikke i semantikken af clausen.
2. Hvis en clause har mere end 3 literals gør følgende: $F = (l_1 \vee l_2 \vee A)...$ hvor A er en OR af 2 eller flere literals. Vi indfører nu variabel y og laver nu $F' = (l_1 \vee l_2 \vee y)(\neg y \vee A)$. Sådan fortsætter vi til A består af en OR af kun 2 literals, da vi så er færdige.

2.8 2SAT $\in$ P

2SAT algoritme:
Konstruer implikationsgraf ud fra en given 2SAT formel: Lav 2n noder, en node for hver variabel og dens negering. Tilføj derefter 2 kanter, $(\neg a, b)$ og $(\neg b, a)$ for hver clause $(a \vee b)$. Find stærke sammenhængskomponenter i grafen (cykler) i

lineær tid. Tjek vha bfs/dfs for hver variabel om der er en cykel hvor både x og $\neg x$ indgår. Hvis sådan en variabel eksisterer, så er 2sat formlen unsatisfiable. I alt kan dette gøres i polynomiel tid.



2.9 3SAT $\leqslant$ MAX2SAT

Givet en 3SAT formel konstruerer vi en MAX2SAT formel som følger:
For hver clause $C_i = (\alpha \vee \beta \vee \gamma)$, tilføj følgende 10 clauses:

$$(\alpha)(\beta)(\gamma)(c_i)$$
$$(\neg\alpha \vee \neg\beta)(\neg\beta \vee \neg\gamma)(\neg\alpha \vee \neg\gamma)$$
$$(\alpha \vee \neg c_i)(\beta \vee \neg c_i)(\gamma \vee \neg c_i)$$

Så hvis 3SAT formlen har m clauses, så har r(x) 10m clauses. The MAX2SAT goal K is equal to 7m.

Enhver tilfredsstillende assignment for C_i vil tilfredsstille præcis 7 af de tilsvarende 2sat clauses i r(x). Hvis alle literals i 3SAT clausen evaluerer til true, så er hele den første og sidste række tilfredsstillet hvis vi sætter c_i til true. Hvis 2 ud af 3 literals i 3SAT formlen er true, så er 2 clauses tilfredsstillet i hver række, og vi får den 7. uanset sandhedsværdien af c_i. Hvis kun en literal er tilfredsstillet er hele 2. række og 1 clause fra første og sidste række tilfredsstillet. Ved at sætte c_i til false opnås 7 clauses.
Hvis alle literals evaluerer til false kan max 6 clauses i vores konstruktion tilfredsstilles, hvilket er som ønsket.

3 NP-complete graph problems.

3.1 Beslutningsproblemer og stand-in sprog

3.2 P, NP, NPC

3.3 Reduktioner

3.4 3SAT $\leqslant$ HAMILTONIAN PATH

Theorem 9.7: HAMILTONIAN PATH is NP-Complete

We prove this by reducing from 3SAT, which we know is NP-complete.

We construct a graph G such that G has a Hamiltonian path if and only if the input 3sat formula has a satisfying assignment.

For each variable we add a choice gadget and join them. For each clause we add a triangle where each edge corresponds to a literal in the clause. For each edge in the triangle, we add a consistency gadget from the edge, to the edge in the corresponding choice gadget. All nodes in all triangles belong to a clique together with the edge following the last choice gadget, and some other node which is the only one adjacent to the last edge in our path (see figure 6.)

The important thing is that all triangle have at least one edge visited (or actually the 4 nodes on the edges) by the time we get to the clique member after the last choice gadget. If this is not the case then we cannot visit all nodes in a triangle without visiting one of the clique members twice. Recall that edges in the triangles represent literals in the clause - the edges we visit via the choice gadgets are the one for which the corresponding literal make the clause true, and so if no edge is visited, then the clause is false. This is consistent with the fact that we cannot make a hamiltonian cycle.

Figure 5 show exactly what happens with the consistency gadget. Our path goes through nodes 1,2,3,4... and leaves at 12. It is clear that none of the black nodes are visited.

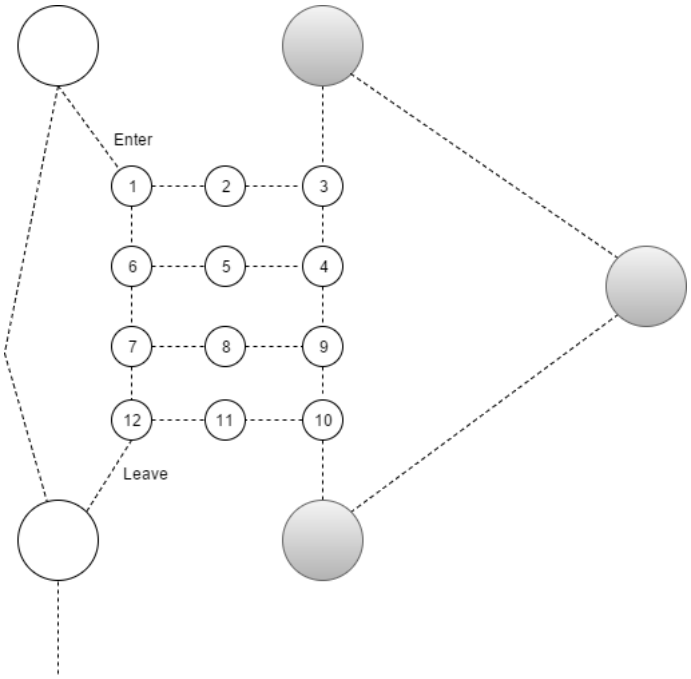


Figure 5: Zooming in on the consistency gadget between the choice gadget and the triangle.

Example:
Given a 3SAT formula $f = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
We can construct a graph similar to the one in figure 6. For the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 0$, we can obtain a hamiltonian path by starting at 1, and follwing the pink line to 2.

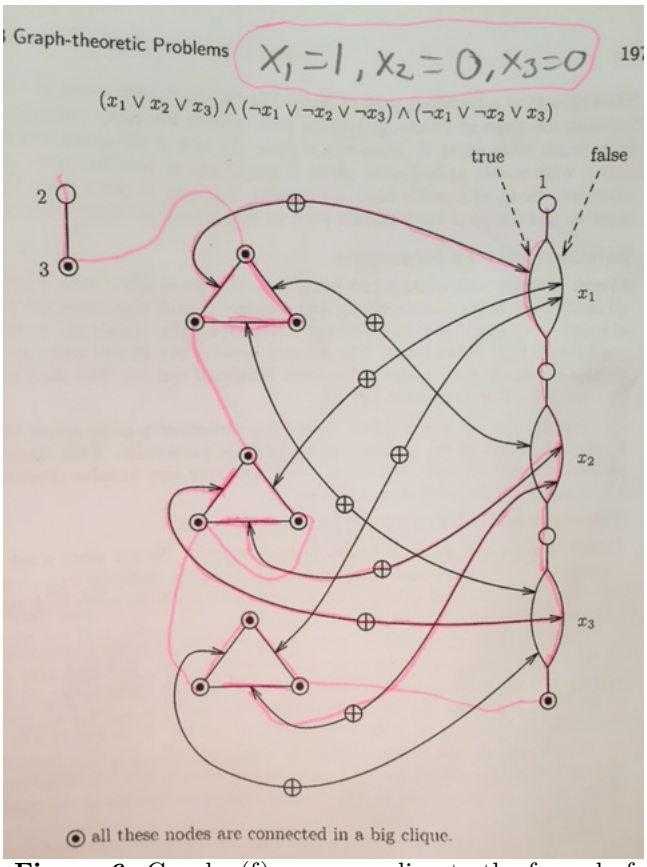


Figure 6: Graph r(f) corresponding to the formula f

3.5 HAMILTONIAN PATH $\leqslant$ TSP

//TODO

3.6 3SAT $\leqslant$ INDEPENDENT SET

Theorem 9.4: INDEPENDENT SET is NP-Complete

We prove this by reducing from 3SAT, which we know is NP-Complete.

For this reduction we need a gadget, the triangle. The logic behind this is that, if a graph contains a triangle, then at most one of the nodes can be in the independent set. We restrict the class of graphs we consider, to graphs whose nodes can be partitioned in m disjoint triangles. This ensures that an independent set can contain at most m nodes.

For each of the m clauses in our input CNF formula, we create a triangle where the nodes are labeled with the literals of the clause. Next, we add an edge between two nodes in different triangles if and only if the nodes correspond to opposite literals. Adding these edges between opposite literals, ensures that we cannot pick both (as it would not be an independent set). The reduction is completed by setting the independent set goal K = m.

Example:
Given a 3CNF formula: $f = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$
We construct the graph shown in figure 1.

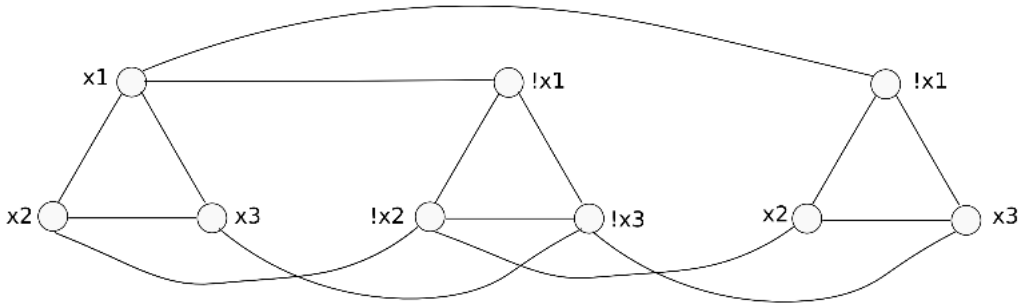


Figure 1: Graph r(f) corresponding to the formula f

Analysis
For each clause, we just create a triangle. Then we scan every pair of edges and add an edge if the labels are opposite - so the reduction is clearly polynomial.

$\Longrightarrow$:

Given a satisfying assignment for the CNF formula f from above, we must show that the graph, r(f), has an indenpendent set of size K=m. The assignment: $x_1 = 0, x_2 = 1, x_3 = 0$ satisfies the formula, and for our independent set of size K, we pick a node in each triangle which make the corresponding clause true. For the truth assignment we chose above, this means x_2 from triangle 1, $\neg x_1$ in triangle 2, $\neg x_1$ in triangle 3. (Note that this is not the only valid set since some clauses are satisfied by more than one literals).

$\Longleftarrow$:
Given an independent set of size K of r(f), we need to show that there is a satisfying assignment for f. Since it is of size K, it must have one vertex from each triangle, and it cannot contain a variable and its negation. Now, for each vertex in the independent set, we look at the label and assign a truth value to the corresponding variable, so the literal of the label become true. If a variable is not restricted to a truth value by any vertex label, we just assign it some value, since all clauses are already satisfied by the assignments to the other variables.

Read more in papadimitriou page 188-190.

3.7 INDEPENDENT SET $\leqslant$ CLIQUE

The independent set problem asks for a maximum set of vertices of size K. So does the CLIQUE problem. While the independent set problem asks for a set S of vertices where there is no edge between any pair $u, v \in S$, the clique problem ask for the "opposite": a set S where any pair $u, v \in S$ DOES have an edge between them. So the reduction is trivial - we obtain a clique of size K when taking the complementary graph of an independent set of size K.

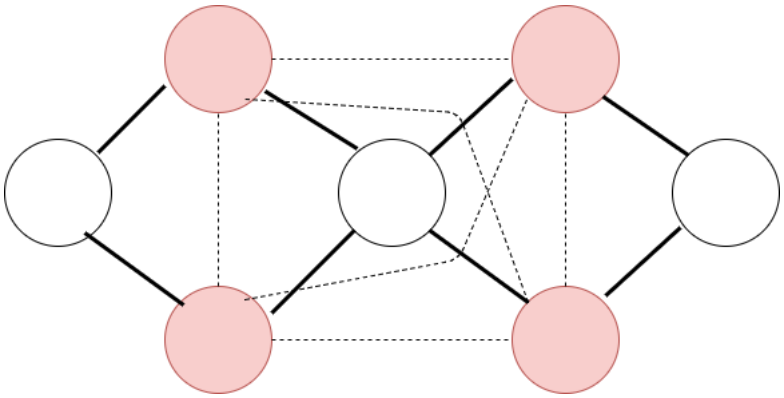


Figure 2: The red nodes are the maximum independent set I - the dotted lines completes the complementary graph of I. A clique of same size.

Read more in papadimitriou page 190.

3.8 INDEPENDENT SET $\leqslant$ VERTEX COVER

While independent set is a maximization problem, vertex cover is a minimization problem. For a graph G = (V,E) and a max independent set $I \subseteq V$, the minimum vertex cover set is just $V - I$.

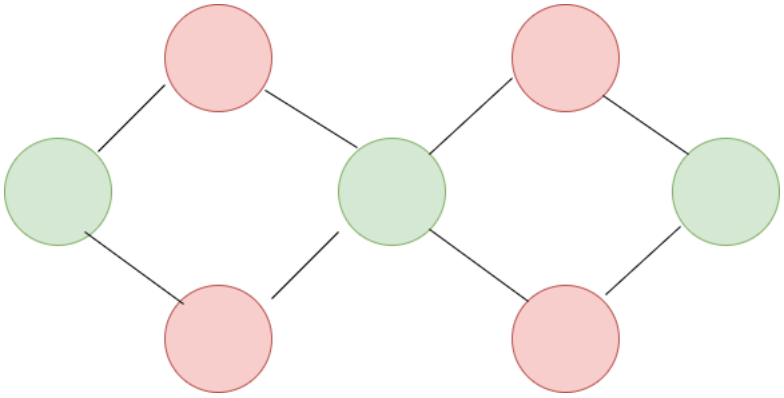


Figure 3: The red node forms the maximum independent set. The green form the minimal vertex cover.

Read more in papadimitriou page 190.

4 NP-complete problems involving sets and numbers.

4.1 Beslutningsproblemer og stand-in sprog

4.2 P, NP, NPC

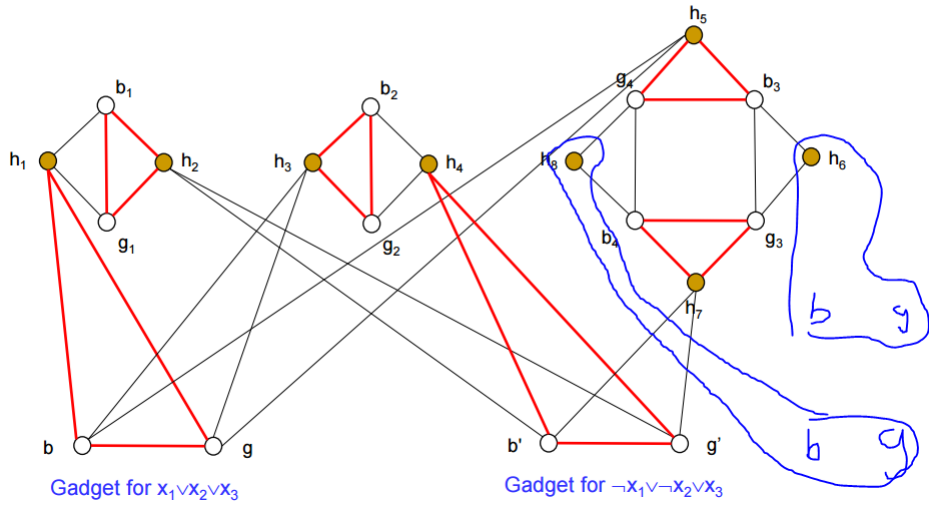
4.3 Reduktioner

4.4 3SAT $\leqslant$ TRIPARTITE MATCHING

For this reduction we use a combined and consistency gadget. We create one for each variable x_i. Each one of such gadget will have k boys, k girls and 2k homes. Where k is equal to the highest occurrence of x_i or $\neg x_i$.

For each clause we add another pa

Example: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$: the matching and the satisfiable assignment $T(x_1)$=true, $T(x_2)$=false, $T(x_3)$=false



4.5 Corollary: EXACT COVER BY 3-SETS, SET COVERING AND SET PACKING ARE NP COMPLETE

TRIPARTITE MATCHING is a special case of EXACT COVER BY 3-SETS. And EXACT COVER BY 3-SETS is a special case of both SET COVER and SET PACKING.

(https://people.eecs.berkeley.edu/ vazirani/algorithms/chap8.pdf)

4.5.1 TRIPARTITE MATCHING $\leqslant$ EXACT COVER BY 3-SETS

Given a TRIPARTITE MATCHING instance: let the EXCB3-S subsets be the same as in the TRIPARTITE MATCHING instance. The EXCB3-S universe is obtained by the union of B G and H.

Think of EXCB3-S as kind of the same problem but where you can match 3 Boys, or 2 girls and 1 home etc.

4.5.2 EXACT COVER BY 3-SETS $\leqslant$ SET COVERING

EXACT COVER BY 3-SETS instances are a subset of all SET COVERING instances, so the is easy. Nothing is changed and the Budget B = m from the EXCB3-S instance.

4.5.3 NODE COVER $\leqslant$ SET COVERING

SET COVER is a generalization of NODE COVER. In our construction the universe is all edges in the NODE COVER instance. The subsets are obtained as follows: For each node in the graph, all adjacent edges form a subset.

4.5.4 EXACT COVER BY 3-SETS $\leqslant$ SET PACKING

Set the SET PACKING goal k to be m. Then there is a 3-SET cover of size m iff there is a SET PACKING of size m. (Since the input set is 3m). Rest is unchanged.

4.6 EXACT COVER BY 3-SETS $\leqslant$ KNAPSACK

Once again we will prove a problem NP hard by showing that a special case is NP hard. The special case of KNAPSACK where $v_i = w_i$ and goal $K = W$ is called SUBSET SUM. (see definition in first section).

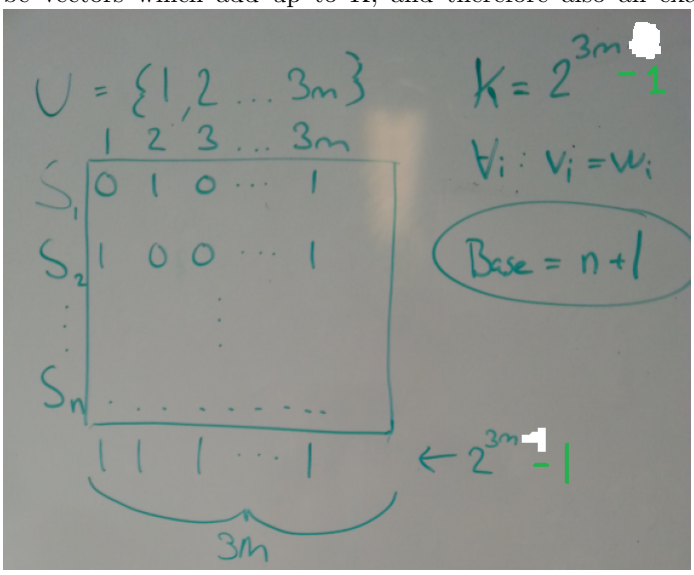
In our construction we think of the 3-sets as binary vectors in $\{0,1\}^{3m}$. And the i'th bit will be 1 iff the i'th element is contained with this particular 3-set. Therefore in each vector, only 3 ones will appear. The vectors can be thought of as binary integers, and set union now resembles integer addition and the goal is to find vectors which sum up to the all 1's vector.

But binary addition has a carry, which could break the bi-implication. For example 0011 + 0101 + 0111 = 1111, but the sets $\{3,4\}$,$\{2,4\}$,$\{2,3,4\}$ are not disjoint, and their union is not $\{1,2,3,4\}$.
To fix this, we think of the numbers in base n+1 instead of 2. since we have n vectors, there can never be a carry.

The weight goal of $2^{3m} - 1$ completes the reduction.

$\Longrightarrow$:
If there is an exact cover by 3-sets, then the bit vectors in our construction corresponding to the exact cover will provide a number equal to the wanted K.

$\Longleftarrow$:
Since we can include every number once, and there is no carry, then there must be vectors which add up to K, and therefore also an exact cover by 3-sets.



5 Approximation algorithms.

5.1 Disposition

1. P, NP, NPC

6 Problemer og flere reduktioner

6.1 INDEPENDENT SET

Given: Graph G = (V,E), target K.
Question: Does there exist $I \subseteq V$ such that $|I| \geqslant K$ and for all $u, v \in I$ we have $(u, v) \notin E$?

6.2 CLIQUE

Given: Graph G = (V,E), target K.
Question: Does there exist $C \subseteq V$ such that $|C| \geqslant K$ and for all $u, v \in C$ we have $(u, v) \in E$?

6.3 VERTEX COVER

Given: Graph G = (V,E), budget B.
Question: Does there exist $C \subseteq V$ such that $|C| \leqslant B$ and for all $(u, v) \in E$ we have $(u \in C \vee v \in C)$?

6.4 MAX CUT

Given: Graph G = (V,E), target K.
Question: Does there exist a cut $(S, V \backslash S)$ of size at least K?

6.5 MAX BISECTION

Given: Graph G = (V,E), target K.
Question: Does there exist a cut $(S, V \backslash S)$ of size at least K, such that $|S| = |V \backslash S|$?

6.6 BISECTION WIDTH

Given: Graph G = (V,E), target K.
Question: Does there exist a cut $(S, V \backslash S)$ of size at most K, such that $|S| = |V \backslash S|$?

6.7 HAMILTONIAN PATH

Given: Graph G = (V,E).
Question: Does G have a path that visits every vertex exactly once?

6.8 TSP

Given: Distance matrix D, target t.
Question: Is there a tour of length at most t that visits every node in the graph defined by D exactly once?

6.9 SET COVERING

Given: A set U(universe). A collection of subsets $S_1, S_2, ..., S_n \subseteq U$
Question: Does there exist B subsets whose union is U?

6.10 EXACT COVER BY 3-SETS

Given: A set U of size 3m. A collection of subsets $S_1, S_2, ..., S_n \subseteq U$ of size three.
Question: Does there exist m subsets which cover U exactly?

6.11 TRIPARTITE MATCHING

Given: Three sets B, G, H of size n. A collection of triples $T \subseteq B \times G \times H$.
Question: Does there exist n triples such that every element of B, G and H is contained in exactly one of these triples?

6.12 SET PACKING

Given: A set U. A collection of subsets $S_1, S_2, ..., S_n \subseteq U$. A goal K.
Question: Does there exist k pairwise disjoint subsets?

6.13 KNAPSACK

Given: N items. Item i has value v_i and weight w_i, both positive integers. A max weight W. And a goal K.
Question: is there a subset of the N items such that the total weight is at most W and that the total value is at least K?

6.14 SUBSET SUM (SPECIAL CASE OF KNAPSACK)

In this version of knapsack an items value is equal to its weight, and the goal K is equal to the max weight W. So, we are given a set of N integers and an integer K and ask if there is a subset of the given integers that adds up to K.

6.15 BIN PACKING

Given: N positive integers and two more integers C(capacity) and B(number of bins).
Question: can these numbers be partitioned into B subsets, each of which has a total sum of at most C?

6.16 NAESAT $\leqslant$ MAX CUT

Theorem 9.5: MAX CUT is NP-Complete

We prove this by reducing from NAESAT, which we know is NP-complete.

We construct a graph G = (V,E), and a goal K such that there is a way to separate the nodes og G in to two sets S and V-S with K or more edges going from one set to the other, if and only if there is a truth assignment for our NAESAT formula f which satisfies f. We allow more than one edge between two vertices, and each such edge contributes one to the cut.

Our input formula has m clauses and n variables, and we constuct a graph with 2n nodes - one for each variable and it's negation. For each clause, we add edges to our graph to form triangles. If two literals of a clause are the same, for example $(x \vee y \vee y)$, (which is equivalent to $(x \vee y)$), we just add two edges between the nodes representing the two destinct literals. The point of the triangles and the two edges between same literals, is that for both constructs, the cut is always two. Now we add an edge between variables and their negation, and set our target K = n + 2m. **The n is for cutting the horizontal edges, and 2m is because we need to cut each triangle (each clause) - if we don't, then it is possible for all literals in a single clause to be true (and then it's not naesat))**. This completes the reduction.

Example:
Given a NAESAT formula: $f = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
We can construct the graph shown in figure 4. The green edges come from the first clause, red the second clause, and blue the third clause.

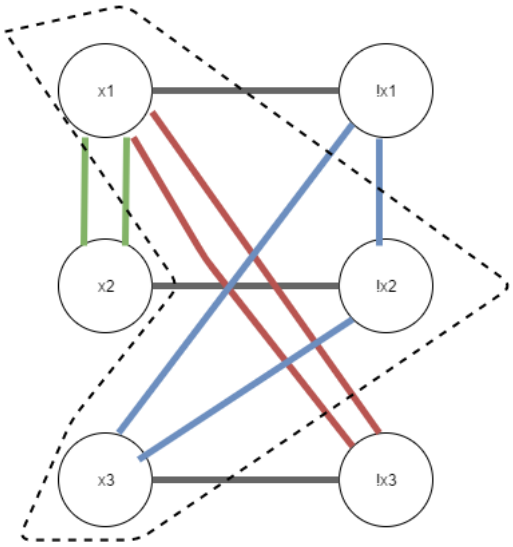


Figure 4: Graph r(f) corresponding to the formula f

Analysis:
For each variable we add two nodes and one edge between them. For each clause we add at most 3 edges. So the reduction is polynomial.

$\Longrightarrow$:
Given a satisfying naesat assignment for f, we must show that r(f) has a cut of size at least K. $x_1 = 1, x_2 = 0, x_3 = 1$ is a satisfying assignment for f. We have 3 variables and 3 clauses, so the target K is $2 * 3 + 3 = 9$. The cut of 9 is obtained by cutting the vertices within the dotted line in the example above.

$\Longleftarrow$:
Given a cut of size at least K of r(f), we must show that there is a satisfying naesat assignment for x. Here we just take the truth values of the nodes in our cut set.

Read more in papadimitriou page 191-192.