Coping with NP-completeness: Special Cases

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Advanced Algorithms and Complexity
Data Structures and Algorithms

The fact that a problem is **NP**-complete does not exclude an efficient algorithm for special cases of the problem.

Outline

1. 2-Satisfiability
2. Independent Sets in Trees

This part

- Striking connection between strongly connected components of a graph and formulas in 2-CNF
- A linear time algorithm for 2-SAT

2-Satisfiability (2-SAT)

Input: A set of clauses, each containing at most two literals (that is, a 2-CNF formula).

Output: Find a satisfying assignment (if exists).

Example

- $(x \vee y)(\overline{z})(z \vee \overline{x})$ is satisfied by $x = 0, y = 1, z = 0$

Example

- $(x \vee y)(\overline{z})(z \vee \overline{x})$ is satisfied by $x = 0, y = 1, z = 0$
- $(x \vee y)(\overline{z})(z \vee \overline{x})(\overline{y})$ is unsatisfiable

Example

- $(x \vee y)(\overline{z})(z \vee \overline{x})$ is satisfied by $x = 0, y = 1, z = 0$
- $(x \vee y)(\overline{z})(z \vee \overline{x})(\overline{y})$ is unsatisfiable
- $(x \vee y)(x \vee \overline{y})(\overline{x} \vee y)(\overline{x} \vee \overline{y})$ is unsatisfiable

- Consider a clause $(\ell_1 \vee \ell_2)$

- Consider a clause $(\ell_1 \vee \ell_2)$
- Essentially, it says that ℓ_1 and ℓ_2 cannot be both equal to 0

- Consider a clause $(\ell_1 \vee \ell_2)$
- Essentially, it says that ℓ_1 and ℓ_2 cannot be both equal to 0
- In other words, if $\ell_1 = 0$, then $\ell_2 = 1$ and if $\ell_2 = 0$, then $\ell_1 = 1$

Definition

Implication is a binary logical operation denoted by $\Rightarrow$ and defined by the following truth table:

x	y	$x \Rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

Definition

For a 2-CNF formula, its implication graph is constructed as follows:

- for each variable x, introduce two vertices labeled by x and $\overline{x}$;
- for each 2-clause $(\ell_1 \vee \ell_2)$, introduce two directed edges $\overline{\ell}_1 \to \ell_2$ and $\overline{\ell}_2 \to \ell_1$
- for each 1-clause (ℓ), introduce an edge $\overline{\ell} \to \ell$

Definition

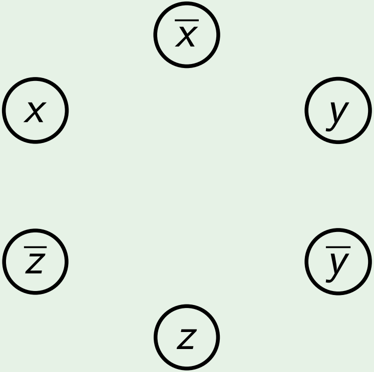
For a 2-CNF formula, its implication graph is constructed as follows:

- for each variable x, introduce two vertices labeled by x and $\overline{x}$;
- for each 2-clause $(\ell_1 \vee \ell_2)$, introduce two directed edges $\overline{\ell}_1 \to \ell_2$ and $\overline{\ell}_2 \to \ell_1$
- for each 1-clause (ℓ), introduce an edge $\overline{\ell} \to \ell$

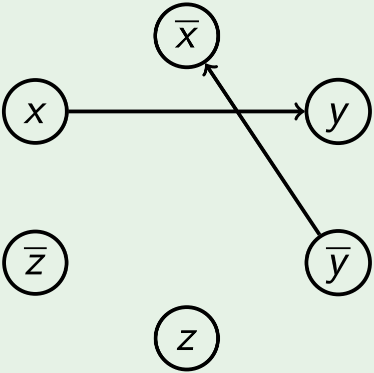
Encodes all implications imposed by the formula.

$$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$$

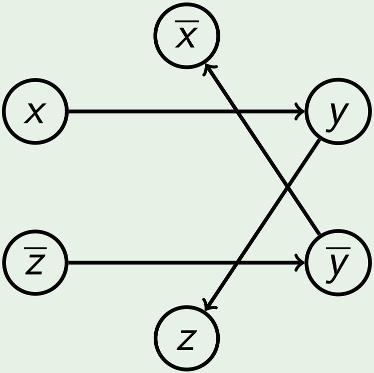
$$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$$



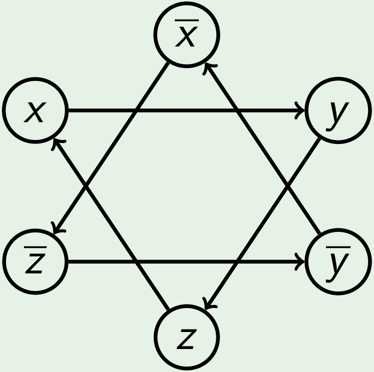
$$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$$



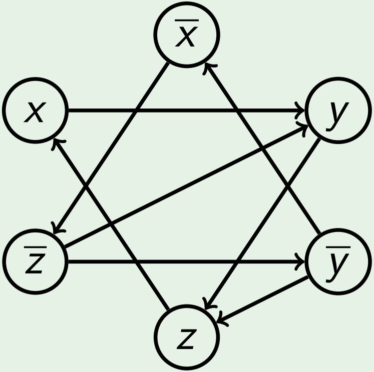
$$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$$



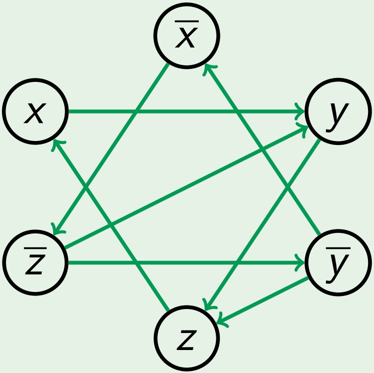
$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$



$$(\overline{x} \lor y)(\overline{y} \lor z)(x \lor \overline{z})(z \lor y)$$

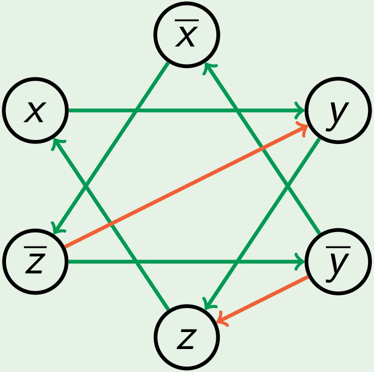


$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$



$x = 1, y = 1, z = 1$

$$(\overline{x} \vee y)(\overline{y} \vee z)(x \vee \overline{z})(z \vee y)$$



$$x = 0, y = 0, z = 0$$

Thus, our goal is to assign truth values to the variables so that each edge in the implication graph is "satisfied", that is, there is no edge from 1 to 0.

Skew-Symmetry

- The graph is skew-symmetric: if there is an edge $\ell_1 \rightarrow \ell_2$, then there is an edge $\overline{\ell}_2 \rightarrow \overline{\ell}_1$

Skew-Symmetry

- The graph is skew-symmetric: if there is an edge $\ell_1 \rightarrow \ell_2$, then there is an edge $\overline{\ell}_2 \rightarrow \overline{\ell}_1$
- This generalizes to paths: if there is a path from ℓ_1 to ℓ_2, then there is a path from $\overline{\ell}_2$ to $\overline{\ell}_1$

Transitivity

Lemma

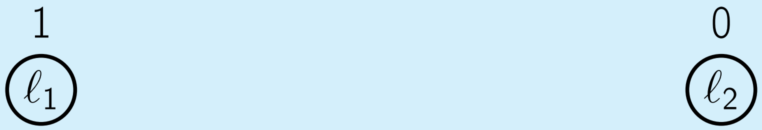
If all the edges are satisfied by an assignment and there is a path from ℓ_1 to ℓ_2, then it cannot be the case that $\ell_1 = 1$ and $\ell_2 = 0$.

Transitivity

Lemma

If all the edges are satisfied by an assignment and there is a path from ℓ_1 to ℓ_2, then it cannot be the case that $\ell_1 = 1$ and $\ell_2 = 0$.

Proof

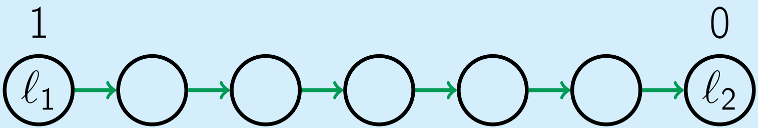


Transitivity

Lemma

If all the edges are satisfied by an assignment and there is a path from ℓ_1 to ℓ_2, then it cannot be the case that $\ell_1 = 1$ and $\ell_2 = 0$.

Proof

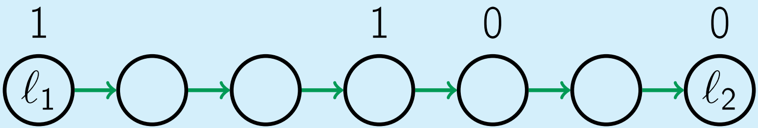


Transitivity

Lemma

If all the edges are satisfied by an assignment and there is a path from ℓ_1 to ℓ_2, then it cannot be the case that $\ell_1 = 1$ and $\ell_2 = 0$.

Proof

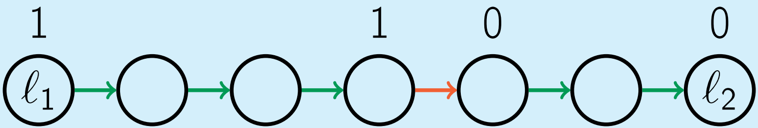


Transitivity

Lemma

If all the edges are satisfied by an assignment and there is a path from ℓ_1 to ℓ_2, then it cannot be the case that $\ell_1 = 1$ and $\ell_2 = 0$.

Proof



Strongly Connected Components

- All variables lying in the same SCC of the implication graph should be assigned the same value

Strongly Connected Components

- All variables lying in the same SCC of the implication graph should be assigned the same value
- In particular, if a SCC contains a variable together with its negation, then the formula is unsatisfiable

Strongly Connected Components

- All variables lying in the same SCC of the implication graph should be assigned the same value
- In particular, if a SCC contains a variable together with its negation, then the formula is unsatisfiable
- It turns out that otherwise the formula is satisfiable!

2SAT(2-CNF F)

```
construct the implication graph G
find SCC's of G
for all variables x:
  if x and x̄ lie in the same SCC of G:
    return "unsatisfiable"
find a topological ordering of SCC's
for all SCC's C in reverse order:
  if literals of C are not assigned yet:
    set all of them to 1
    set their negations to 0
return the satisfying assignment
```

2SAT(2-CNF F)

```
construct the implication graph G
find SCC's of G
for all variables x:
  if x and x̄ lie in the same SCC of G:
    return "unsatisfiable"
find a topological ordering of SCC's
for all SCC's C in reverse order:
  if literals of C are not assigned yet:
    set all of them to 1
    set their negations to 0
return the satisfying assignment
```

Running time: $O(|F|)$

Lemma

The algorithm `2SAT` is correct.

Proof

- When a literal is set to 1, all the literals that are reachable from it have already been set to 1 (since we process SCC's in reverse topological order).

Lemma

The algorithm `2SAT` is correct.

Proof

- When a literal is set to 1, all the literals that are reachable from it have already been set to 1 (since we process SCC's in reverse topological order).
- When a literal is set to 0, all the literals it is reachable from have already been set to 0 (by skew-symmetry). □

Outline

1. 2-Satisfiability

2. Independent Sets in Trees

Planning a company party

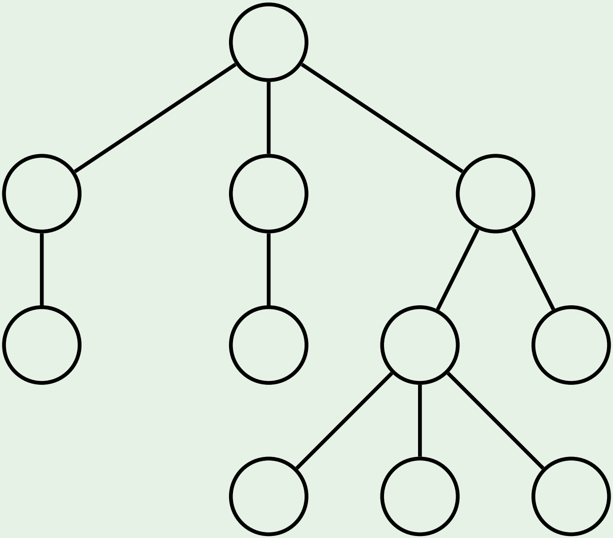
You are organizing a company party. You would like to invite as many people as possible with a single constraint: no person should attend a party with his or her direct boss.

Maximum independent set in a tree

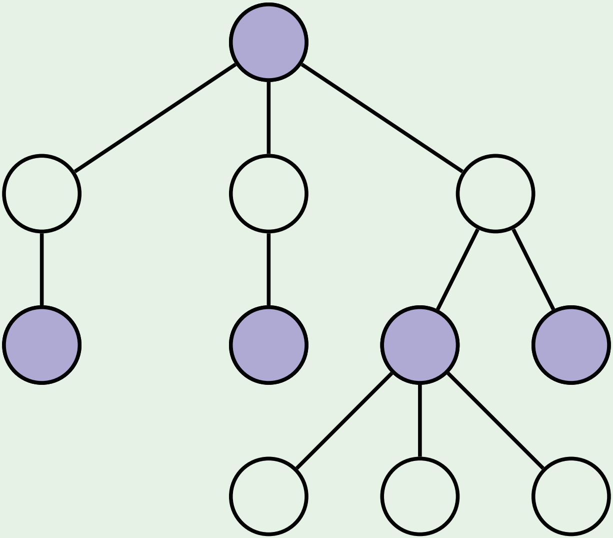
Input: A tree.

Output: An independent set (i.e., a subset of vertices no two of which are adjacent) of maximum size.

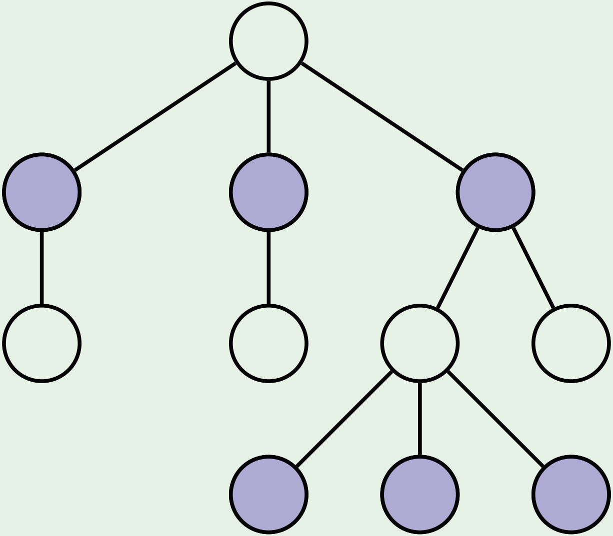
Example



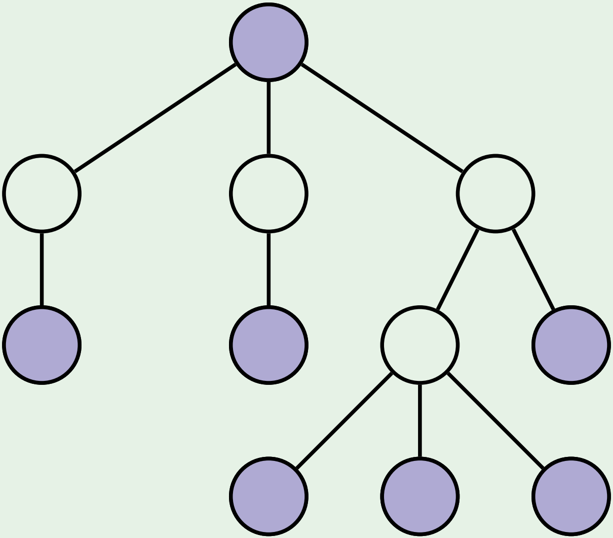
Example



Example

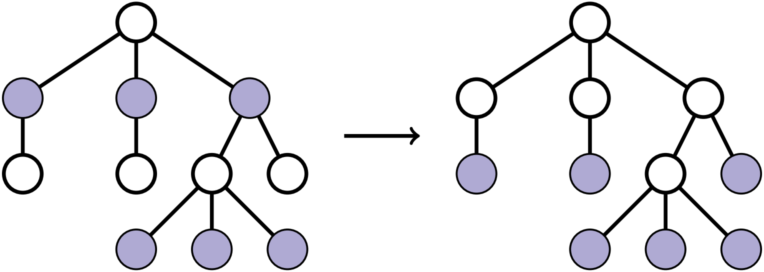


Example



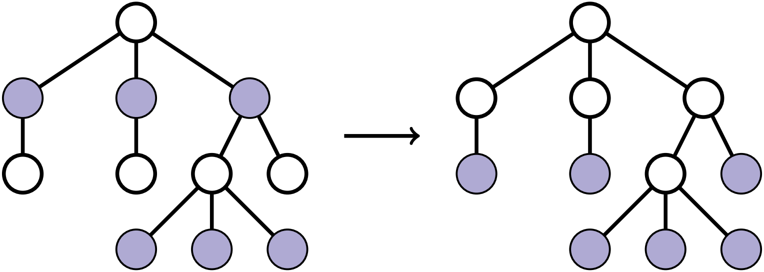
Safe move

For any leaf, there exists an optimal solution including this leaf.



Safe move

For any leaf, there exists an optimal solution including this leaf.



It is safe to take all the leaves.

PartyGreedy(T)

```
while T is not empty:
  take all the leaves to the solution
  remove them and their parents from T
return the constructed solution
```

PartyGreedy(T)

```
while T is not empty:
  take all the leaves to the solution
  remove them and their parents from T
return the constructed solution
```

Running time: $O(|T|)$ (for each vertex, maintain the number of its children).

Planning a company party

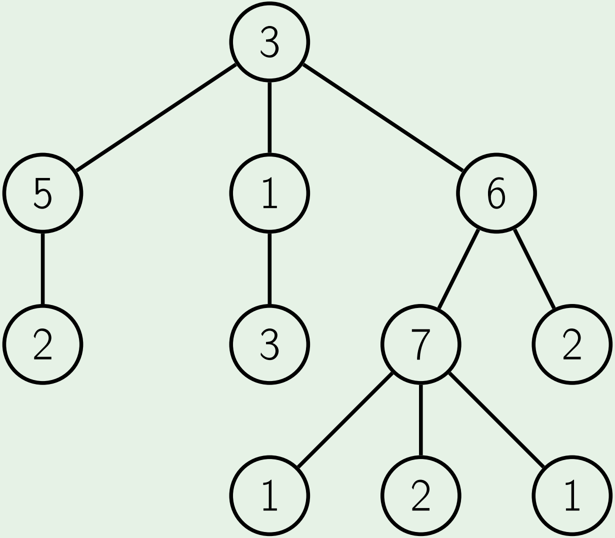
You are organizing a company party again. However this time, instead of maximizing the number of attendees, you would like to maximize the total fun factor.

Maximum weighted independent set in trees

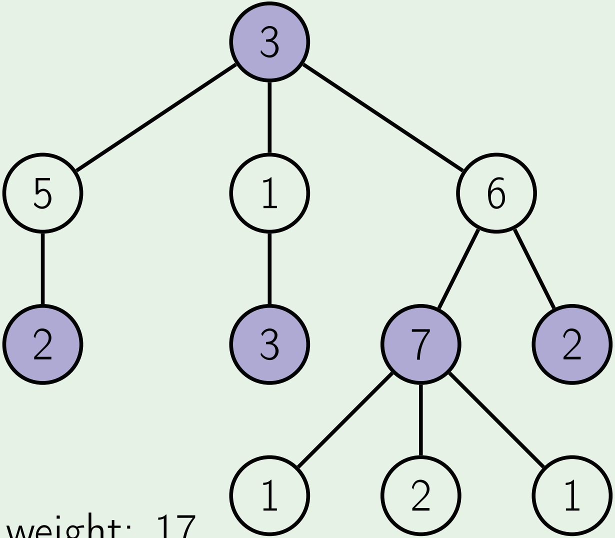
Input: A tree T with weights on vertices.

Output: An independent set (i.e., a subset of vertices no two of which are adjacent) of maximum total weight.

Example

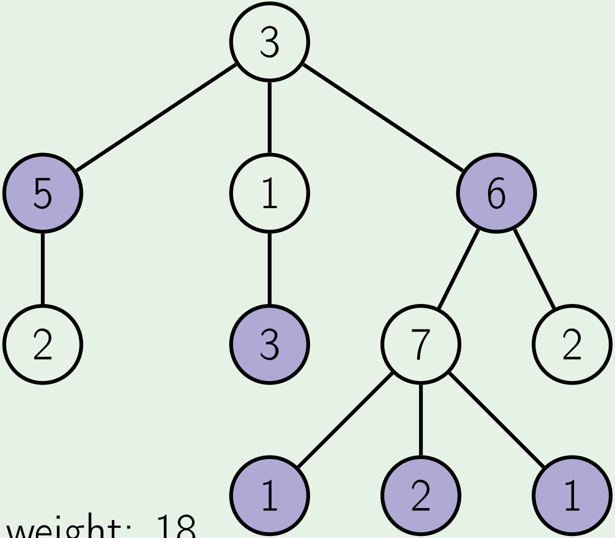


Example



total weight: 17

Example



total weight: 18

Subproblems

- $D(v)$ is the maximum weight of an independent set in a subtree rooted at v

Subproblems

- $D(v)$ is the maximum weight of an independent set in a subtree rooted at v
- Recurrence relation: $D(v)$ is

$$\max \left\{ w(v) + \sum_{\substack{\text{grandchildren} \\ w \text{ of } v}} D(w), \sum_{\substack{\text{children} \\ w \text{ of } v}} D(w) \right\}$$

Function FunParty(v)

```
if D(v) = ∞:
  if v has no children:
    D(v) ← w(v)
```

Function FunParty(v)

```
if D(v) = ∞:
  if v has no children:
    D(v) ← w(v)
  else:
    m_1 ← w(v)
    for all children u of v:
      for all children w of u:
        m_1 ← m_1 + FunParty(w)
```

Function FunParty(v)

```
if D(v) = ∞:
  if v has no children:
    D(v) ← w(v)
  else:
    m₁ ← w(v)
    for all children u of v:
      for all children w of u:
        m₁ ← m₁ + FunParty(w)
    m₀ ← 0
    for all children u of v:
      m₀ ← m₀ + FunParty(u)
```

Function FunParty(v)

```
if D(v) = ∞:
  if v has no children:
    D(v) ← w(v)
  else:
    m1 ← w(v)
    for all children u of v:
      for all children w of u:
        m1 ← m1 + FunParty(w)
    m0 ← 0
    for all children u of v:
      m0 ← m0 + FunParty(u)
  D(v) ← max(m1, m0)
return D(v)
```

Example