Fixed-Parameter Algorithm for 2-CNF Deletion Problem

Igor Razgon
Computer Science Department
University College Cork
Ireland
A brief introduction to the area of fixed-parameter algorithms
Fixed-parameter algorithms as a way of coping with NP-hardness

Fixed-parameter algorithms allow to solve hard optimization problems:
- exactly,
- in a low-polynomial time.

Of course, they are exponential in the worst case, but:
- the degree of the exponent does not depend on the size of input but on an additional parameter $k$ associated with the problem
- in real-world instances the value of the parameter is frequently very small
Definition of a fixed-parameter algorithm

Given an intractable problem with the input size $n$ and a parameter $k$.

A fixed-parameter algorithm is an algorithm that solves this problem in time $O(f(k) \cdot n^c)$, where $f(k)$ is an exponential function of $k$, $c$ is a constant independent on $k$. 
Example: a fixed-parameter algorithm for the Vertex Cover Problem

**Vertex Cover problem:**
given a graph $G$, find the smallest Vertex Cover (VC), i.e. a set of vertices incident to all the edges of $G$.

Being parameterized by the size of VC, the problem asks: given a non-negative integer $k$, find out whether $G$ has a VC of size at most $k$. 
Example: a parameterized algorithm for the Vertex Cover Problem (cont.)

FindVC(G,k)
If G has no edges then return ‘YES’
If k=0 then return ‘NO’
Select an edge \{u, v\} of G
If FindVC(G\u,k-1) returns ‘YES’ or
   FindVC(G\v,k-1) returns ‘YES’ then
   Return ‘YES’
Else
   Return ‘NO’
Example: a fixed-parameter algorithm for the Vertex Cover Problem (further cont.)

- The recursive applications of \textit{FindVC} can be organized into a search tree.

\begin{center}
\begin{tikzpicture}
    \node (root) at (0,0) {$\text{FindVC}(G,k)$};
    \node (left) at (-1,-2) {$\text{FindVC}(G\setminus u,k-1)$};
    \node (right) at (1,-2) {$\text{FindVC}(G\setminus v,k-1)$};
    \draw[->] (root) -- (left);
    \draw[->] (root) -- (right);
\end{tikzpicture}
\end{center}

- The height of the tree is at most \( k \). Each non-leaf node has 2 children. Hence the search tree has \( O(2^k) \) nodes. The complexity of \textit{FindVC} is \( O(2^kn) \).

- A better algorithm for the VC problem takes \( O(1.3^k+n) \). It works in a reasonable time for \( k=60 \) and a huge \( n \).

- It is much better then to explore all subsets of \( k \) vertices.
Fixed-parameter tractability

• A problem that can be solved by a fixed-parameter algorithm is called fixed-parameter tractable (FPT).

• As we have seen, the VC problem parameterized by the size of the output is FPT.

• Another example: many graph-theoretical problems (e.g. Clique) are FPT being parameterized by the treewidth of the underlying graph.
Applications

Fixed-parameter algorithms can be applied in the areas where the problems are associated with parameters that are very small in practice. Such areas include:

- Bioinformatics
- Networks Design
- Computer Security
- Machine Learning
- Artificial Intelligence
Fixed-parameter tractability vs. intractability

Parameterized Independent Set Problem
Given a graph $G$ and a parameter $k$, find out whether $G$ has an independent set (a set of mutually non-adjacent vertices) of size at least $k$.

A simple method of solving the problem.
Select a vertex $v$. Select into the independent set either $v$ or one of neighbours of $v$ and apply the algorithm recursively to the corresponding residual graph with decreasing the parameter by 1.

Runtime analysis
The algorithm creates a search tree of height $k$, but the number of children of a node is the number of neighbours of the corresponding vertex plus one. For dense graphs the number of nodes of the tree may be as large as $O(n^k)$. Thus this algorithm is not a fixed-parameter one. This suggests that there might be no fixed-parameter algorithm solving the problem.
Fixed-parameter tractability vs. intractability (cont.)

A stronger evidence that the Independent Set Problem is not FPT: If it is FPT then the widely believed Exponential Time Hypothesis fails (i.e., 3-SAT, Independent Set, and many other problems can be solved in a subexponential time).

The question of classification:
given an intractable problem, find out whether this problem is FPT or probably not.
The 2-CNF deletion problem
Terminology

An example of a 2-CNF formula: \((X_1 \lor X_2) (\neg X_2 \lor \neg X_3) (X_3 \lor \neg X_1)\).

- The clauses of the formula are \((X_1 \lor X_2)\), \((\neg X_2 \lor \neg X_3)\), and \((X_3 \lor \neg X_1)\).
- The variables of the formula are: \(X_1\), \(X_2\), and \(X_3\).
- The literals of the formula are: \(X_1\), \(X_2\), \(X_3\), \(\neg X_1\), \(\neg X_2\), \(\neg X_3\).
- A literal included in a clause satisfies this clause.
- An assignment of a formula is a set of its literals, exactly one per variable.
- A 2-CNF formula \(F\) is satisfied by an assignment \(P\) if each clause of \(F\) is satisfied by at least one literal of \(P\).
- A satisfying assignment of the formula in the above example is \(\{X_1, \neg X_2, X_3\}\).
Parameterized 2-CNF deletion problem (2-CNF-DEL)

**Input:** 2-CNF formula $F$ and a parameter $k$.

**Output:** ‘YES’ if there is a set of at most $k$ clauses whose removal makes the resulting formula satisfiable.
Two equivalent problems

1. **Input:** Graph $G$, parameter $k$.  
   **Output:** ‘YES’ if $G$ has a VC greater than the maximum matching of $G$ by at most $k$, ‘NO’ otherwise.

   *This is a more general parameterization of VC that by the output size*

2. **Input:** CNF formula $F$ (not necessary 2-CNF!), parameter $k$. 
   **Output:** ‘YES’ if there are $k$ variables such after their removal from $F$, the resulting formula is RENAMABLE HORN, ‘NO’ otherwise.

   *This problem has applications in the design of SAT solvers.*
Fixed-parameter tractability of the 2-CNF-DEL problem

• The question about the fixed-parameter tractability of the 2-CNF–DEL problem was first asked by Mahajan and Raman in JALG 31(2) pp. 335-354, 1999 (the preprint appeared two years earlier in ECCC 4(33), 1997).

• Since then this question acquired reputation of one of the central challenges in the design of fixed parameter algorithms (see Niedermeier, “Invitation to fixed-parameter algorithms, volume 31, Oxford University press, page 277).

• The fixed-parameter tractability of this problem has been confirmed by Igor Razgon and Barry O’Sullivan in “Almost 2-SAT is fixed parameter tractable”, ICALP 2008.
A fixed-parameter algorithm for the 2-CNF-DEL problem
The general idea of the algorithm

The basic procedure:

Problem with the second branch:

the parameter does not decrease

the height of the search tree is not guaranteed to be bounded by a function of $k$

the algorithm is not necessary a fixed-parameter one.
The general idea of the algorithm (cont.)

A way to fix the problem: introducing a \textit{polynomially computable lower bound on the optimal solution size} and recognizing 3 cases.

1. The lower bound is greater than $k$. ‘NO’ is returned immediately

2. Forbidding the selected clause increases the lower bound.

The gap between the parameter and the lower bound decreases on both branches $\Rightarrow$ the height of the search tree depends on $k$. 

19
The general idea of the algorithm (further cont.)

3. Forbidding the selected clause does not increase the lower bound.

**Theorem:**
Forbidding the selected clause does not increase optimal solution size → the selected clause can be forbidden *without any branching.*

On each path from the root to a leaf of the search tree, the number of nodes with 2 or more children depends on $k$ → the number of leaves of the search tree depends on $k$ → the algorithm is a fixed-parameter one.
**Auxiliary problem**

The above strategy is in fact applied to an auxiliary problem, not to the 2-CNF-DEL problem directly. We define the auxiliary problem and show that if this problem is FPT then the 2-CNF-DEL problem is FPT as well.

A 2-CNF formula $F$ is *satisfiable w.r.t.* a set of literals $S$ if there is a satisfying assignment $P$ of $F$ such that $S$ is a subset of $P$.

For example, $(X_1 \lor X_2)(\neg X_2 \lor \neg X_1)$ is satisfiable w.r.t. $\{X_1\}$ while $(\neg X_1 \lor X_2)(\neg X_2 \lor \neg X_1)$ is not.

**Problem AUX**

**Input:** $(F,S,I,k)$, where $F$ is a 2-CNF formula, $S$ is a set of literals such that $F$ is satisfiable w.r.t. $S$, $I$ is a literal of $F$, $k$ is the parameter.

**Output:** ‘YES’ if there is a set of at most $k$ clauses of $F$ whose removal makes resulting formula satisfiable w.r.t. $S \cup \{I\}$; ‘NO’ otherwise.
Theorem.
If problem AUX is FPT then the 2-CNF-DEL problem is FPT as well.

In the proof we show that the 2-CNF-DEL problem can be solved by making $O(3^k \cdot m)$ calls to a procedure solving problem AUX, where $m$ is the number of clauses of $F$.

Problem AUX as a graph separation problem

We show that problem AUX can be represented as a separation problem on the implication graph of a 2-CNF formula.

The implication graph $D(F)$ of a 2-CNF formula $F$ is a directed graph whose set of vertices corresponds to the set of literals of $F$ and $(X_1, X_2)$ is an arc of $D(F)$ iff $(\neg X_1 \lor X_2)$ is a clause of $F$.

No bijection between clauses and arcs: an arc of $D(F)$ corresponds to exactly one clause of $F$ while a clause of $F$ generally corresponds to two different arcs of $F$.

In the above example, the additional arc associated with clause $(\neg X_1 \lor X_2)$ is $(\neg X_2, \neg X_1)$.

Let $A$ and $B$ be two sets of vertices of $D(F)$. A set $C$ of clauses of $F$ is an $(A, B)$-separator if removal of all the arcs corresponding to the clauses of $C$ breaks all the paths from $A$ to $B$ in $D(F)$.
Problem AUX as a graph separation problem

**Theorem**

Let \((F,S,l,k)\) be an instance of problem AUX. This is a ‘YES’ instance if and only if \(F\) has an \((S \cup \{l\},\{\neg l\})\) separator of size of size at most \(k\).

The proof is similar to the proof of the unsatisfiability criterion of a 2-CNF formula (see, for example, “Computational Complexity” by Papadimitriou).
We introduce a polynomially computable lower bound on the size of \((S \cup \{l\}, \{\neg l\})\) separator. This lower bound is necessary for implementation of the general algorithmic scheme.

The smallest possible size of an \((S, \{\neg l\})\) separator is a lower bound on the size of \((S \cup \{l\}, \{\neg l\})\) separator.

**Theorem.**
A smallest \((S, \{\neg l\})\) separator is polynomially computable.

**Proof sketch.**

\(S\) is satisfiable w.r.t. \(F\) \(\rightarrow\) two arcs corresponding to the same clause cannot be simultaneously reachable from \(S\) \(\rightarrow\) there is a bijection between the edges reachable from \(S\) and the corresponding clauses \(\rightarrow\) the smallest \((S, \{\neg l\})\) separator can be computed by a network flow algorithm.
**Clauses forbidden for removal**

We describe instances of problem AUX to which the algorithm is recursively applied if the selected clauses is forbidden to be removed.

Assume that \((F,S,l,k)\) is a ‘YES’ instance of problem AUX and clause \((X_1 \lor X_2)\) is forbidden to be removed. Then the resulting formula must be satisfiable either w.r.t. \(S \cup \{X_1\}\) or w.r.t. \(S \cup \{X_2\}\).

Therefore forbidding the clause generally requires two recursive applications: one to \((F,S \cup \{X_1\},l,k)\), the other to \((F,S \cup \{X_2\},l,k)\).

**Remark.** The algorithm is applied to instance \((F,S \cup \{X_i\},l,k)\) only if \(F\) is satisfiable w.r.t. \(S \cup \{X_i\}\). Otherwise, the respective branch is omitted.
Neutral literals

A literal $l'$ of a 2-CNF formula $F$ is a neutral literal of instance $(F,S,l,k)$ of problem AUX if the size of a smallest $(S, \{\neg l\})$-separator is the same as the size of a smallest $(S U \{l'\}, \{\neg l\})$-separator.

**Theorem.**
If $l'$ is a neutral literal of $(F,S,l,k)$ and $(F,S,l,k)$ is a ‘YES’ instance of problem AUX then $(F,S U \{l'\}, l,k)$ is a ‘YES’ instance as well.

**Corollary.**
If $(X_1 v X_2)$ is a clause of $F$ and $X_1$ is a neutral literal of $(F,S,l,k)$ then without any branching clause $(X_1 v X_2)$ can be forbidden and algorithm can recursively apply to $(F,S U \{X_1\}, l,k)$. 
The algorithm

\textit{SolveAUX}(F,S,l,k)

\textbf{If} $F$ is satisfiable w.r.t. $S \cup \{l\}$ then return ‘YES’

Let $LB$ be the size of a smallest $(S,\{\neg l\})$-separator

\textbf{If} $LB > k$ \textbf{then} return ‘NO’

Select a clause $C = (X_1 \lor X_2)$

\textbf{If} some $X_i$ is a neutral literal of $(F,S,l,k)$ \textbf{then}

\hspace{1em} Return \textit{SolveAUX}(F,S \cup \{X_i\},l,k)

\textbf{If} \textit{SolveAUX}(F \setminus C,S,l,k-1) \textbf{ returns ‘YES’ or}

\hspace{1em} \textit{SolveAUX}(F,S \cup \{X_1\},l,k) \textbf{ returns ‘YES’ or}

\hspace{1em} \textit{SolveAUX}(F,S \cup \{X_2\},l,k) \textbf{ returns ‘YES’ then}

\hspace{1em} Return ‘YES’

\textbf{Else} return ‘NO’
Runtime evaluation of \textit{SolveAUX}

Theorem.
Each path from the root to a leaf in the search tree contains at most $2k$ branching nodes.

Proof sketch.
• Each node is associated with measure $2k-LB$.
• The measure of the root is at most $2k$.
• A node with measure $0$ is a leaf.
• The measure of any child of a branching node is smaller than the measure of this node itself.

Clause removal decreases the parameter by 1 and decreases $LB$ by at most 1 $\Rightarrow$ the measure of the corresponding child is at most $2(k-1)-(LB-1) < 2k-LB$.
Clause forbidding increases $LB$ by 1 $\Rightarrow$ the respective measure becomes $2k-(LB+1) < 2k-LB$. 


Corollary.
The search tree has at most $9^k$ leaves.

This corollary immediately follows from the last theorem, taking into account that each branching node has at most 3 branches. A more careful evaluation allows to prove that the number of leaves of the search tree can be bounded by $5^k$. Taking into account the polynomial factors, the runtime of $\text{SolveAUX}(F,S,I,k)$ is $O(5^kkm^2)$, where $m$ is the number of clauses of $F$. Taking into account that $O(3^km)$ calls to $\text{SolveAUX}$ are needed to solve the 2-CNF-DEL problem, we get the final theorem.

Theorem.
2-CNF-DEL problem can be solved in time $O(15^kkm^3)$
Summary

• Fixed-parameter algorithms are techniques of coping with NP-hardness that are useful in situations where problems are associated with parameters that are very small in practice. Problems that can be solved by fixed-parameter algorithms are called fixed-parameter tractable (FPT).

• 2-CNF deletion problem asks whether at most \( k \) clauses can be removed from a 2-CNF formula to make it satisfiable. There are a number of reasons while it is worthwhile to solve this problem by a fixed-parameter algorithm. Nevertheless, the status of fixed-parameter tractability of this problem had been open for more than 10 years.

• We have shown that this problem is FPT.